VAX DOCUMENT User Manual, Volume 1

Order Number: AA–JT84B–TE

July 1988

This manual describes the VAX DOCUMENT product for the new user. It includes a full description of the doctype-independent tags used in all types of VAX DOCUMENT documents.

Revision/Update Information:	This revised manual supersedes the *VAX DOCUMENT User Manual, Volume 1 Version 1.0* (Order Number AA–JT84A–TE).
Operating System and Version:	VMS Version 4.7 or higher MicroVMS Version 4.7 or higher
Software Version:	VAX DOCUMENT Version 1.1

digital equipment corporation
maynard, massachusetts

First printing, July 1987
Revised, July 1988

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

This document was prepared using VAX DOCUMENT, Version 1.0

Contents

Preface

Document Structure

This manual describes the VAX DOCUMENT generic markup system and the tags used to create all VAX DOCUMENT types of documentation.

- Chapter 1 provides an introduction of the VAX DOCUMENT system.
- Chapters 2 and 3 describe how to create input files and use basic text elements within those input files.
- Chapter 4 discusses processing of input files, including how to build a book.
- Chapter 5 describes some methods to troubleshoot input files.
- Chapter 6 explains how a user can cross-reference symbol-names throughout files and books.
- Chapter 7 describes how to create a table of contents, an index, or a master index.
- Chapter 8 explains the coding of some special VAX DOCUMENT features.
- Chapter 9 is the reference section of the book, containing an alphabetic listing of all the SDML tags that are used in any doctype. Each tag is explained in full, with examples to illustrate the correct coding.
- Appendix A explains the DOCUMENT command and the command's parameters and qualifiers.
- Appendix B is an overview of the Language-Sensitive Editor that can be used with VAX DOCUMENT.
- Appendix C lists and explains the error messages users could see when processing source files.
- Appendix D is a table of all SDML tags (both global and doctype-specific). It also indicates in which doctype the tag is valid and summarizes the tag's use.

The VAX DOCUMENT User Manual, Volume 2 provides a description of the tags used in specific doctypes.

The VAX DOCUMENT Design Samples manual provides information on each of the doctype designs available in VAX DOCUMENT and provides samples of each design.

Intended Audience

This book is intended for writers, editors, and general users who wish to produce technical manuals, brochures, or even business letters or overhead slides using VAX DOCUMENT. Familiarity with a text editor is presumed, as is a basic knowledge of the VMS operating system.

Associated Documents

The reader of this manual should be familiar with:

- *Step-by-Step: Writing with VAX DOCUMENT*

Other books in the documentation set for VAX DOCUMENT Version 1.1 which describe the more technical aspects of the product are:

- *VAX DOCUMENT User Manual, Volume 2*
- *VAX DOCUMENT Doctype Designer's Guide*
- *VAX DOCUMENT Installation Guide*
- *VAX DOCUMENT Design Samples*

Conventions

Within both the *VAX DOCUMENT User Manuals, Volumes 1 and 2,* capitalized words within syntax statements indicate specific commands or keywords to be entered. Lowercase words indicate user-specified parameters or arguments. Optional items within syntax statements are enclosed in brackets.

Within the reference chapters in the *VAX DOCUMENT User Manuals, Volumes 1 and 2,* the discussion of each tag follows a fixed order. First, the name of the tag is followed by a brief overview that describes the purpose of the tag. Following the overview is a format section that displays the syntax of the tag: any optional or required arguments and the information required, any related tags, any restrictions on the use of the tag, and any required terminators, if needed.

The category of "related tag" has been defined broadly. A tag is related to the tag under discussion if one of the following criteria is met:

- It is required for use of the tag under discussion.
- It marks a text element of the same kind as the tag under discussion.
- It is commonly used with the tag under discussion.

Following the format section is an optional description section. The description expands the overview and presents more detailed information on the use of the tag. Not every tag requires a description section.

The discussion of a tag concludes with at least one example, or a reference to an example. The example shows how the tag is used in an SDML file and what the formatted result is when the file is processed for printing.

Output examples may vary depending on the doctype you processed the example under or on whether any doctype modifications have been made to your local installation of VAX DOCUMENT. Each output example is introduced by a form of the sentence "This example may produce the following output" to remind you that the output examples may vary.

The following table lists some of the typographical conventions used in this manual.

Convention	Meaning
RET	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, you must press the RETURN key after entering a command or responding to a prompt.)
$ TYPE MYFILE.DAT . . .	In examples, a vertical ellipsis represents the omission of data that the system would display in response to a command or of data a user would enter.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

New and Changed Features

The following table lists the new and changed SDML tags for VAX DOCUMENT Version 1.1 (see Chapter 9 for a complete description of any of these tags):

Type	SDML Tag	Description
New	<DELAYED>	A doctype-independent tag that allows you to specify text that contains SDML tags in an argument to another tag.
Changed	<FIGURE_FILE>	This tag now accepts a keyword argument of PS rather than POST to specify a POSTSCRIPT® device, and a keyword argument of LINE rather than LINE_PRINTER to specify a monospaced line printer.
New	<FILE_SPEC>	A doctype-independent tag that allows you to specify a file specification containing angle brackets as an argument to an SDML tag without VAX DOCUMENT interpreting that file specification as an SDML tag.
Changed	<ICON_FILE>	This tag now accepts a keyword argument of PS rather than POST to specify a POSTSCRIPT device, and a keyword argument of LINE rather than LINE_PRINTER to specify a monospaced line printer.
New	<SET_FIGURE_FILE_SPACING_DEFAULT>	A doctype-independent tag that allows you to override the default amount of blank space that appears before and after an included graphics file.
New	<SET_TABLE_ROW_BREAK_DEFAULT>	A doctype-independent tag that allows you to override the default value for a multipage table's first valid break.
Changed	<TABLE_ATTRIBUTES>	This tag now accepts two new keyword arguments. The CONTROLLED keyword specifies that the breaking of a table will be under explicit control of the <TABLE_ROW_BREAK> tags only. The SINGLE_SPACED keyword specifies that the rows in a table are to be single-spaced.

®PostScript is a registered trademark of Adobe Systems, Inc.

The following table lists the new and changed qualifiers and parameters for VAX DOCUMENT Version 1.1 (see Appendix A for a complete description of any of these parameters or qualifiers):

Type	Qualifier or Parameter	Description
New	/OUTPUT	Allows the user to redirect the VAX DOCUMENT output file to a file other than the default.
Changed	/DEVICE_CONVERTER	The MULTINATIONAL_CHARACTER_SET keyword is no longer valid; by default the DEC multinational character set is used for line printer type devices. The new FALLBACK keyword allows users to disable the DEC multinational character set for a seven-bit monospaced device.
Changed	LINE_PRINTER	LINE_PRINTER is no longer a default destination keyword for VAX DOCUMENT Version 1.1; the new default keyword for that type of destination is LINE.
Changed	LN03_LASER_PRINTER	LN03_LASER_PRINTER is no longer a default destination keyword for VAX DOCUMENT Version 1.1; the new default keyword for that type of destination is LN03.
Changed	/MASTER_INDEX	Master index processing no longer terminates at the text processing stage; it now runs to completion and generates a printable file. Also all the keywords accepted by the /INDEX qualifier are now accepted by the /MASTER_INDEX qualifier.
Changed	POSTSCRIPT	POSTSCRIPT is no longer a default destination keyword for VAX DOCUMENT Version 1.1; the new default keyword for that type of destination is PS. In addition the default file types associated with the POSTSCRIPT keyword have been changed from DVI_POST to DVI_PS, and POST to PS.

1 Introduction to VAX DOCUMENT

The VAX DOCUMENT system for producing technical documentation is designed to fully automate the creation of documentation from generically coded input files. VAX DOCUMENT supports the creation of technical documentation from the earliest stages of a document's planning to the final copy that emerges ready for production.

The basis of the VAX DOCUMENT system is an integrated series of software processors that convert generically coded source files into formatted final output. This integrated system allows users to perform the following functions:

- Write and maintain files for a document
- Produce output for a wide range of devices
- Correct errors and incorporate changes into files with a text editor

Although it uses a powerful, high-level language for text processing, VAX DOCUMENT is not difficult to learn. One or two trial runs of an input file through the processors is all you need do to get a good grasp of the system. It is recommended that you work through the exercises in *Step-by-Step: Writing with VAX DOCUMENT* before you begin more complex production tasks.

This chapter gives a brief overview of the VAX DOCUMENT system and introduces concepts you need to understand and use the system.

1.1 The VAX DOCUMENT Documentation Set

The books in the VAX DOCUMENT documentation set are written for three audiences: a writer or editor who will use VAX DOCUMENT to create documentation, a system manager who will install and maintain the system, and a doctype designer who will create new doctypes for local use. The documentation set includes the following books:

- *Step-by-Step: Writing with VAX DOCUMENT*
- *VAX DOCUMENT User Manual, Volume 1*
- *VAX DOCUMENT User Manual, Volume 2*
- *VAX DOCUMENT Design Samples*
- *VAX DOCUMENT Doctype Designer's Guide*

New users of VAX DOCUMENT might want to start with *Step-by-Step: Writing with VAX DOCUMENT* for a beginning tutorial of how to use the system. The *VAX DOCUMENT User Manual, Volume 1* contains more detailed information, including a reference chapter of the global tags (tags available in all the doctypes). The *VAX DOCUMENT User Manual, Volume 2* contains detailed information on the doctype-specific tags.

Doctype designers might want to start with *Step-by-Step: Writing with VAX DOCUMENT* and then progress to the *VAX DOCUMENT Doctype Designer's Guide*. The *VAX DOCUMENT Design Samples* shows the various doctype designs available in each of the VAX DOCUMENT doctypes.

1.2 Features of VAX DOCUMENT

The following are some of the most important features of VAX DOCUMENT.

Text Formatting

- Adjustable text alignment
- All POSTSCRIPT® and LN03 fonts support the full DIGITAL Multinational Character Set
- Automatic page numbering
- Automatic headers/footers
- Controllable hyphenation
- Multicolumn output

Publishing

- A sophisticated table utility for creating tables in multicolumn formats
- Automated cross-referencing
- Automated collation of table of contents, chapters, and headings
- Automated generation of indexes and master indexes
- Automated generation of table of contents
- Availability of 15 basic document designs
- Batch or interactive processing support
- Generation of footnotes in several formats

Supported Output Devices

- DIGITAL's POSTSCRIPT devices, such as the PRINTSERVER 40 and the LN03R SCRIPTPRINTER
- Line printers
- LN03 and LN03 PLUS laser printers
- Terminal display, to read output on a terminal screen
- VMS Mail format, to read from within the VMS Mail Utility

® PostScript is a registered trademark of Adobe Systems, Inc.

1.3 Using a Generic Markup Language

The language that is used to mark up a file for processing through VAX DOCUMENT is referred to as generic because it is used, unchanged, to produce any type of document. Using a generic markup system requires a different perspective on writing than is needed when you use a formatting language (such as DIGITAL Standard Runoff). It takes time to become accustomed to thinking solely in terms of content rather than format. You need not think in terms of margins, spacing, and other formatting characteristics. Instead, when using a markup system like VAX DOCUMENT, you must think in terms of *text elements*. This means that for each individual element of text, such as the start of a paragraph, an emphasized word, a heading, and so on, you identify that text element for the processor by *tagging* it.

For example, compare a sample of DIGITAL Standard Runoff (DSR) code to the code of VAX DOCUMENT. DSR is coded as follows:

```
.b.lm5.nf.nj
Here is a sample of text that could be
processed by a text processor.
.lm0.f.j.
```

This method requires users to remember the current margins, the amount to indent the text, and so on, each time they type an example. Using SDML, the underlying format is controlled for consistency by the document type (doctype). The example is coded with tags that generically label the kind of text they are formatting:

```
<CODE_EXAMPLE>
Here is a sample of text that could be
processed by a text processor.
<ENDCODE_EXAMPLE>
```

VAX DOCUMENT automatically specifies the correct formatting instructions for a chosen doctype, freeing the writer from the task of formatting text and allowing the writer to concentrate on the task of writing.

The language that you use to tag a file, the Standard DIGITAL Markup Language (SDML), contains a full set of tags for naming all the text elements commonly found in technical documentation. Tagging text elements is discussed in detail in Chapters 2 and 4. For now, just realize that you do not need to control the format as you write. The formatting is taken care of automatically during processing.

All of your writing is done in an input file, which is also called an SDML file because of its file extension of .SDML. When you create a new file that will eventually be processed through VAX DOCUMENT, always give the file an .SDML extension.

1.4 Introduction to the DOCUMENT Command Line

One command, typed at the DCL prompt, is used to process any type of VAX DOCUMENT file:

```
$ DOCUMENT input-file-spec doctype.design destination
```

The DOCUMENT command requires three parameters:

- *Input-file-spec*

  Specifies the input file for VAX DOCUMENT. This file is by default an SDML file containing SDML tags. However, it may also be one of the intermediate files generated by VAX DOCUMENT.

- *Doctype*

  Specifies the document type keyword for which the input file should be processed. This keyword specifies the kind of document to be created (a letter, a software manual, a journal article, and so on). A doctype can comprise several different designs; in that case, the design keyword is specified after the doctype keyword, separated from it by a period.

- *Destination*

  Specifies the final processing destination for the input file. This keyword typically specifies a format used by a printer, but may specify formats for terminals or the VMS Mail Utility (MAIL).

You can also add qualifiers to the DOCUMENT command to create indexes, master indexes, tables of contents, and to modify the default processing of your input file. The DOCUMENT command, parameters, and qualifiers, are explained in Appendix A. Chapter 4 discusses various processing tasks accomplished by using the command line.

1.5 Types of Documents

One of most powerful features of VAX DOCUMENT is its ability to format the same input file into a variety of document types (*doctypes*). The standard doctypes offered with the product cover a range of documentation needs. For VAX DOCUMENT Version 1.1, there are seven supported doctypes (see Table 1–1). Also, most doctypes offer more than one design. The doctypes are explained fully in the *VAX DOCUMENT User Manual, Volume 2*.

In Table 1–1, the first design listed for each doctype is the default design. It is designated when you type the doctype with no design on the DOCUMENT command line. For example, specifying MANUAL on the command line defaults to MANUAL. REFERENCE.

Table 1–1 VAX DOCUMENT Doctypes

Doctype	Design	Description
ARTICLE	ARTICLE	Creates an article format in an $8\frac{1}{2} \times 11$ inch format, with two columns for text.
LETTER	LETTER	Creates memos and letters.
MANUAL	MANUAL.GUIDE	Creates a user manual in a 7×9 inch trim size with numbered headings; it is intended for chapter-oriented tutorial material.
	MANUAL.PRIMER	Creates a user manual in a 7×9 inch trim size with unnumbered headings; it is intended for chapter-oriented primer material.
	MANUAL.REFERENCE	Creates a user manual in an $8\frac{1}{2} \times 11$ inch trim size with numbered headings; it is intended for reference material.
MILSPEC	MILSPEC	Offers a full implementation of the United States Department of Defense Military Specification Standard MIL-STD-490A and MIL-STD-2167. It accepts the full range of VAX DOCUMENT global tags, with the exception of the tags used to create part pages.
OVERHEADS	OVERHEADS	Creates a page with an $8\frac{1}{2} \times 11$ inch trim size. It can be used to create slides that fit on overhead projectors, or figures that fit into an $8\frac{1}{2} \times 11$ inch notebook.
	OVERHEADS.35MM	Creates a $6\frac{1}{2} \times 5\frac{1}{2}$ inch format suitable for camera-ready copy for making 35MM slides.
REPORT	REPORT	Creates general purpose documents such as reports and formal outlines.
	REPORT.TWOCOL	Creates a two column format of the REPORT doctype.
SOFTWARE	SOFTWARE.BROCHURE	Creates a brochure in a 7×9 inch trim size with unnumbered headings and sequential page numbers; it is intended for nonchapter-oriented material.
	SOFTWARE.GUIDE	Creates a users' guide in a 7×9 inch trim size with numbered headings.
	SOFTWARE.HANDBOOK	Creates a handbook in a 7×9 inch trim size with numbered headings.
	SOFTWARE.POCKET_ REFERENCE	Creates a pocket reference in a $5\frac{1}{2} \times 7$ inch trim size with numbered headings.
	SOFTWARE.REFERENCE	Creates a reference manual in an $8\frac{1}{2} \times 11$ inch trim size with numbered headings. It provides various formats and tags for describing computer software. It is equally well-suited to describing information in a tutorial or in a reference format, and contains tags customized for use in describing software in either of these formats.
	SOFTWARE.SPECIFICATION	Creates a specification in an $8\frac{1}{2} \times 11$ inch trim size with numbered headings.

1.6 Overview of the VAX DOCUMENT System

Several integrated software tools form the components of the VAX DOCUMENT system.

- The Standard DIGITAL Markup Language (SDML), used for coding the document files as they are written, is a generic, high-level, device-independent markup language. It is not concerned with formatting; the information it conveys is information about the logical structure of a document.
- Three processors work on the file when you execute the DOCUMENT command. These processors work sequentially, checking and converting the generically marked text through intermediate files on its way to final printed form. Each file is given the same name as the original source file but is suffixed with the file extension specific to that processor. Table 1–2 lists the file extensions generated by each processor.

1.6.1 How VAX DOCUMENT Processes a File

When you execute the DOCUMENT command, a file is run through three processes:

1. Tag translation
2. Text formatting
3. Device conversion

These three processes transform your input file into formatted copy. During the processing of a file, you can observe the following sequence of events on your terminal (or, after processing, in the log file if the file was processed in BATCH mode).

First, the tag translator translates the tags in the input file. As the tag translator processes your file, you might see error messages on the terminal. For example, if you misspell a tag name or fail to use an ending parenthesis, an error message is displayed. Processing continues so as to find as many errors as possible. If you have consistently misspelled a tag, you receive a message only for the first occurrence of that error, then a final message announces how many undefined tags the tag translator saw. The tag translator's output is a file with the same name as the input file but a file type named TEX (for example, *filename.TEX*).

The text formatter reads the TEX file and formats the pages of output. This processor might also issue error messages. The text formatter creates a file suffixed with the file type extension that reflects the chosen output device. If you specify on the command line that you want the file processed for the LN03 printer, the text in the file is proportionately spaced and the DVI file is named DVI_LN03. For a POSTSCRIPT device, the file is named DVI_PS. For a line printer, a terminal, or the VMS Mail Utility, the DVI file is named DVI_LINE.

The device conversion program runs next and creates a device specific file, with the file type of either LN03, LINE, PS, TERM, or TXT. This file reflects the destination that you specified on the command line. In particular, if you specified LN03 or PS the text in the file will be proportionally spaced, whereas if you specified LINE, the text will be monospaced.

The device-specific file is ready for printing (or reading on an ASCII terminal or mailing through the VMS Mail Utility). The file is printed automatically if it has been run through the three processors, unless you specified /NOPRINT on the command line. The file output by the device converter is printed by default on the device associated with the destination on the command line.

The processors are summarized in Table 1–2.

Table 1–2 VAX DOCUMENT Processors

Processor	File Types Created	Description
The Tag Translator	TEX	A macro processor that translates the source files into files used by the text formatter. The tag translator makes two passes over the source file, first checking for invalid tags and then translating the tags.
The Text Formatter	DVI_LN03, DVI_LINE, DVI_PS	The text formatter reads the TEX file and formats the pages of output. This processor also searches for errors, but only for errors in the format, not in text elements or tags. Like the tag translator, it may issue error messages. It creates a DVI file, suffixed by the destination that you specified on the command line. The DVI_LINE file is produced for the destinations LINE, TERM, and TXT. As the text formatter makes up a page, it automatically formats all page parameters. For example, it considers the best arrangement of paragraphs and text elements and appends to the page the appropriate running head and folio.
The Device Converter	LN03, LINE, PS, TERM, or TXT	The device conversion program runs after the text formatter, and reads the .DVI_*device* file created by the text formatter. Whereas the text formatter has already made the page and line breaking decisions, based on the group of fonts in a given doctype, the device converter makes the formatted file into a printable file. These final output files are ready to be printed, read on an ASCII terminal, or mailed through the VMS Mail Utility. After processing, a printable file is automatically printed unless /NOPRINT was specified on the command line.

1.6.2 Informational Messages

Each of the processors displays an informational message when the DOCUMENT command is used interactively, as shown in the following example:

```
❶ $ DOCU myreport REPORT LN03

  %DOC-I-IDENT, VAX DOCUMENT V1.1
```

```
❷ [ T a g   T r a n s l a t i o n ]...
  %TAG-I-DEFSLOADD, End of Loading of Tag Definitions
  %TAG-I-ENDPASS_1, End of first pass over the input
❸ [T e x t   F o r m a t t i n g ]...
  %TEX-I-PAGESOUT, 17 pages written.
  -TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYREPORT.DVI_LN03'
❹ [ D e v i c e   C o n v e r s i o n ]...
  %DVC-I-PAGESOUT, 18 pages written to file:
     DUA1:[DOCFILES]MYREPORT.LN03
❺ [ P r i n t i n g   F i l e ]...

  Job MYREPORT (queue SYS$LN03, entry 832) started on SYS$LN03

  $
```

❶ Identifies the DOCUMENT command line that you type at the DCL prompt. In this example, the DOCUMENT command was issued for a file called MYREPORT.SDML, (the SDML file type is assumed by the tag translator). The file is to be processed for the REPORT doctype, and the output will be printed on the default LN03 laser printer.

Note: On the command line, the command, qualifiers, and keywords can be abbreviated to the shortest unique string. For example, DOCUMENT can be abbreviated to DOCU, and the local destination, LN03_LOCAL_PRINTER, can be abbreviated to LN03.

❷ Identifies the message that indicates that the tag translator is starting to process the file, and will now log its activities. If there are any tag translator errors, they are reported at this point.

❸ Identifies the point when text formatting begins. Any text formatting errors are seen at this point.

❹ Identifies the point when device conversion begins.

❺ Shows the point that the file has started printing on the specified device. Jobs will print by default in both interactive and batch processing.

1.6.3 Doctype-Independent and Doctype-Specific Tags

The majority of the SDML tags used for marking your text are valid in any of the doctypes. These tags are called "doctype-independent," or "global," as they work in all doctypes for which they are appropriate. For example, all doctypes use paragraphs and lists, so <P> and <LIST> are accepted in all doctypes.

A few of the tags that are listed as global in this manual have restricted use, as they are not appropriate in one or more doctypes. For example, the doctypes LETTER or OVERHEADS do not contain front matter, such as a title page or preface, so the front matter tags (accepted in most doctypes) are restricted in these doctypes.

Tags that are used specifically in one doctype only are called "doctype-specific." For example, only a letter contains an address from the person sending the letter, so the doctype-specific tag <FROM_ADDRESS> can be used only in the LETTER doctype.

Appendix D lists all the global and doctype-specific tags. It also identifies the doctype to which each doctype-specific tag belongs.

When first experimenting with the system, process your input file using the MANUAL doctype because it has no restrictions on the use of global tags. Later, you can experiment with the other doctypes, by reprocessing your SDML files and specifying the new doctype on the command line. If the SDML files include tags that are not recognized in the specified doctype, informational messages are issued, and you can either fix the errors or go back to using the MANUAL doctype.

1.6.4 Context-Sensitive Tags

Some tags are used in a group rather than individually. They are grouped according to function and are defined in the context of other tags. Such context-sensitive tags need a tag that enables the rest of the tags in that group. You generally place the tags in the group in a specific order, and terminate the ordered group with a terminating tag. For example, a table begins with a <TABLE> tag and ends with an <ENDTABLE> tag. Between these tags are additional tags that supply information about the number of columns, column widths, column headings; this pattern of tags is the same for every table.

Other examples of global context-sensitive tags include lists, examples, and figures. An example of doctype-specific context-sensitive tags is the group of reference template tags used in the SOFTWARE doctype.

1.7 Processing Books and Elements of Books

You can use VAX DOCUMENT to process files that are individual components of a large book. A book is usually divided into chapters, whereas other types of documents are not chapter-oriented. In a book, each chapter is considered its own *book element*, and is kept in its own file. By setting up a book in this way, you can process the book's chapters either separately or as a whole. The processing of a whole book is called a *bookbuild*.

Processing a document in a nonbook format (not using a bookbuild) is generally useful when you have a smaller document and plan to process the entire document every time you make a change, and you are not concerned with maintaining cross-references in the book. A bookbuild allows you to build a large book and then process elements of it while maintaining cross-references.

To perform a bookbuild, you must create a separate file that contains a list of all the book's elements in the order that they will appear in the book. The separate file profiles the order of the book, from front to back, and hence is called a *profile*. The book's profile becomes the main vehicle used for processing the book.

2 Creating an SDML File

This chapter provides guidelines for creating an SDML file with VAX DOCUMENT. It explains the use of the various types of code in an SDML file, including tags and arguments. The principal rules of coding syntax are also explained. The use of these rules permits VAX DOCUMENT's tag translator to recognize and act on the tags that mark your text.

2.1 Global Tags

In all files generically coded with VAX DOCUMENT, text must be marked, or "tagged," so that the text is formatted correctly when the file is processed. For every *text element* (that is, every individually treated portion of text), you insert a unique ASCII character sequence to identify that type of text element. These character sequences identify a paragraph as a paragraph, a table as a table, and so on.

Tags are used to identify the text elements. A tag is a term or abbreviation enclosed in angle brackets (<...>). When VAX DOCUMENT processes your SDML file, it recognizes the tags, translates them, and formats the text.

There are three rules concerning VAX DOCUMENT use.

RULE #1:

Type each tag as shown, adding no punctuation or spaces. Use angle brackets to enclose tags. The tag name may be in uppercase or lowercase letters, or a combination of both.

Here are some sample tag formats:

Tag Format	Function
<P>	Starts a paragraph
<CHAPTER>	Marks the beginning of a chapter and specifies its title
<TABLE>	Starts a table (like this one)
<NOTE>	Starts text that receives distinctive treatment
<ENDNOTE>	Ends the text that receives distinctive treatment

There are two types of tags: tags that mark the starting point of a text element (for example, <P> , <TABLE> , or <NOTE>) and tags that completely encompass a text element (for example, <HEAD1> or <CHAPTER>). Of the first type, some require explicit termination (for example, <NOTE> requires <ENDNOTE>) and some do not (for example, <P>). All the text that lies between the starting and terminating tags receives distinctive treatment.

All VAX DOCUMENT global tags are listed alphabetically and described in Chapter 9. The description of every tag in that chapter indicates whether that tag requires explicit termination.

2.1.1 Tag Arguments

There are two types of tag *arguments*, those that provide additional characteristics or identify requirements of the text associated with a tag and those that provide actual text. Some tags require one or more arguments, while other tags do not require arguments but can accept them as additional information. Still other tags do not accept any arguments.

An argument is enclosed in parentheses and immediately follows the closing angle bracket of the tag. If you use an argument with a tag that will not accept one, VAX DOCUMENT issues warning messages when you process the file. More than one argument can be included within the parentheses (this is called an *argument list*). A space or an end of line cannot separate the tag from its argument list.

RULE #2:

Supply only those arguments that are permitted by the tag's format. Use parentheses to enclose arguments.

Use a backslash (\) to separate multiple arguments. For example, the <CHAPTER> tag takes two arguments, a chapter title and a symbol name for that title. Therefore, the title for a chapter of a book would be coded with a backslash, as follows:

```
<CHAPTER>(Introduction to Farming\intro_farm)
```

Each tag description in Chapter 9 includes a list of valid or required arguments to that tag. Use of symbol names is discussed in Chapter 6.

2.2 Coding an SDML File

RULE #3:

Tag every text element.

This rule is true whether or not the tag requires an argument list, a terminating tag, or both.

For example, the following text contains two text elements and so requires two tags:

```
<P>Employment opportunities grew throughout the company. If the president
of the company had realized what a <NEWTERM>(matrix) would start
developing . . .
```

In this example, the writer started a new paragraph and so tagged it with the <P> tag. The writer also wanted to identify a new term in this document with some special format, so treated the term as an individual text element and tagged it. The output of this paragraph might appear like this:

```
Employment opportunities grew throughout the company. If the president of
the company had realized what a  matrix would start
developing . . .
```

The three rules concerning the use of VAX DOCUMENT cover the requirements for coding any SDML file. By remembering these rules, you should be able to code your files with few errors. However, there are several special cases to keep in mind when you want to show coding characters, characters used by VAX DOCUMENT internally, in your output. The following discussion explains these unusual cases.

2.3 Special Coding in an SDML File

VAX DOCUMENT treats certain characters and character-strings in a special manner when they are placed in an argument to an SDML tag. Table 2–1 summarizes these characters and how to correctly code them.

Table 2–1 Coding Special Characters

<table>
<tr><th>Character</th><th>Placement of Character</th><th>Correct Coding in Text</th></tr>
<tr><td><text></td><td>Angle brackets enclose a tag name not meant to be translated as a tag.</td><td><LITERAL> (<text>)</td></tr>
<tr><td><P> (</td><td>Opening parenthesis following a tag which accepts no arguments.</td><td>Leave space between the closing angle bracket (>) and the opening parenthesis (() of the tag.</td></tr>
<tr><td>USER$: <SMITH></td><td>Angle brackets enclose a file specification not meant to be translated as a tag in an argument.</td><td><FILE_SPEC> (USER$: <SMITH>)</td></tr>
<tr><td>(</td><td>Unmatched opening parenthesis in an argument.</td><td><OPAREN></td></tr>
<tr><td>)</td><td>Unmatched closing parenthesis in an argument.</td><td><CPAREN></td></tr>
<tr><td>\</td><td>Backslash in an argument.</td><td><BACKSLASH></td></tr>
<tr><td>|</td><td>Vertical bar in an argument.</td><td><VBAR></td></tr>
<tr><td>&</td><td>Ampersand in an argument.</td><td><AMPERSAND></td></tr>
</table>

The tag translator displays a message when you have incorrectly coded any of these special characters in an argument. The correct coding of each of these characters is explained in detail in the following sections.

Two other special cases are also discussed in this section: Section 2.3.6 discusses how to code a hyphen in text, and Section 2.3.7 discusses the usage of tabs in SDML files.

2.3.1 Coding Text That Looks Like a Tag

When a word is surrounded by angle brackets, the tag translator attempts to evaluate it as a tag. If it is a valid tag, it is translated like any other tag; if it is not a valid tag, a warning message is issued when the file is processed, and the word, enclosed in angle brackets, is placed in the output unchanged.

For example, the following text contains the terms <reroute> and <return> , coded as if they are tags:

```
<P>When marking packages for delivery, do not confuse the
<reroute> destination with the <return> destination.
```

During processing, the tag translator tries to translate these terms as tags. It produces two warning messages because <reroute> and <return> are not defined as valid tags.

To avoid this problem, use the <LITERAL> tag, which directs the tag translator to ignore everything (including the angle bracket characters) in its argument. To have <reroute> and <return> appear in the text but not generate warning messages, the above example should be coded like this:

```
<P>When marking packages for delivery, do not confuse the
<LITERAL>(<reroute>) destination with the
<LITERAL>(<return>) destination.
```

Notice that the entire term that you want to include in the text, including the angle brackets, is treated as an argument to the <LITERAL> tag.

Use the <LITERAL> tag to document any characters that are prefixed with angle brackets. For example, the control characters of the ASCII character set commonly are surrounded by angle brackets, such as <STX> or <NAC> . Another case where you might show an angle bracket is in an equation. For instance, in the equation "If M <P then ...," the tag translator tries to define this as part of the paragraph tag (<P>), and issues a warning.

Note: You can place a single angle bracket anywhere in the file, as long as the next few characters do not look like they begin a tag. If they do look like tags to the tag translator but are not valid tags, you will receive a warning message. Coding the angle bracket within a <LITERAL> argument (<LITERAL> (<)) ensures that you will not receive a warning message for placing a single angle bracket in a file.

2.3.2 Coding File Specifications That Include Angle Brackets as Arguments

VAX DOCUMENT provides several tags that allow you to include separate files into your SDML file; such tags require a file specification argument to access an external file. Because the directory portion of a file specification can be delimited with angle brackets, and so resemble a tag in these arguments, such file specifications must receive special treatment.

Use the <FILE_SPEC> tag to designate a file specification that contains angle brackets as the file-spec argument to an SDML file-inclusion tag.

For example, to include the file <SMITH> picture.sixel into a figure and not have VAX DOCUMENT translate <SMITH> as a tag, you would use the <FILE_SPEC> tag as follows:

```
<FIGURE_FILE>(LN03\<FILE_SPEC>(<smith>picture.sixel)\20)
```

The following SDML tags let you include external files, and so are the only tags that might require the use of the <FILE_SPEC> tag in an argument:

- <ELEMENT>
- <EXAMPLE_FILE>
- <FIGURE_FILE>
- <ICON_FILE>
- <INCLUDE>
- <INCLUDES_FILE>
- <TABLE_FILE>

Section 2.3.1 lists the ways in which you can code file specifications that appear in text and not as file-spec arguments to tags.

2.3.3 Coding Opening and Closing Parentheses in an Argument

When the closing angle bracket of a tag is followed immediately by an opening parenthesis, the tag translator assumes that an argument follows. The tag translator reads the argument, looking for a matching closing parenthesis to mark its end. If one of your arguments includes an unmatched closing parenthesis, the parenthesis will be mistaken for the end of the argument.

If an argument needs to include an unmatched opening or closing parenthesis, you can code it with special tags created for this purpose. Use either the <OPAREN> (that is, opening parenthesis) or the <CPAREN> (closing parenthesis) tag. A parenthesis that is coded this way is not matched with its counterpart by the tag translator. It does not disturb the counting of matched pairs of parentheses, nor the recognition of the true end of the argument list. For example:

```
<HEAD1>(A (CLOSE Statement)
```

must be coded as:

```
<HEAD1>(A <OPAREN>CLOSE Statement)
```

Outside of an argument, parentheses cause no problems. For example, the following line is coded correctly:

```
<P>We always use a (CLOSE Statement in the code...
```

You need only use the <OPAREN> and <CPAREN> tags within arguments because parenthesis characters have no special meaning outside of arguments. For more information on <OPAREN> and <CPAREN>, refer to the tag descriptions in Chapter 9.

Note: Open parentheses following tags that do not accept arguments must be preceded by one or more blank spaces. For example, in the following code a space must be placed after the <P> tag, or the text in the parentheses must be placed on a new line.

```
<P>
(See Section X.)
```

2.3.4 Coding a Backslash in an Argument

A backslash is used to separate arguments. If you want to include a backslash character within one of your arguments, you must code it using the <BACKSLASH> tag. Otherwise, the backslash in your argument is interpreted as an argument separator. In effect, the tag translator will make two arguments out of what you perceive as one argument. For example:

```
<P>Insert the part numbered <KEYWORD>(ABC\123) into  . . .
```

The tag translator treats this as two arguments and issues an error message because this tag cannot accept two arguments. To include a backslash that will not be evaluated by VAX DOCUMENT, write the following:

```
<P>Insert the part numbered <KEYWORD>(ABC<BACKSLASH>123)
into  . . .
```

The following is produced:

```
Insert the part numbered ABC\123 into . . .
```

You need to use this special coding only within arguments. For more information on <BACKSLASH>, refer to its description in Chapter 9.

2.3.5 Coding a Vertical Bar or Ampersand in an Argument

The tag translator assigns special meanings to the vertical bar (|) and the ampersand (&) characters, recognizing them as a pair of internal coding characters. When the tag translator is reading an argument list and finds a vertical bar, it starts looking for a matching ampersand.

Because these tags have special meaning within argument lists, you should code the vertical bar character with the <VBAR> tag and the ampersand character with <AMPERSAND> within your arguments. You need do this only within arguments, however, because these characters have no special meaning outside of arguments.

The following example that shows the difference between using an ampersand in a tag argument and using it in text:

```
<HEAD3>(The Ampersand (<AMPERSAND>) Character)

<P>The ampersand character (&) is used to continue a BASIC
statement on the next line. . . .
```

Within the argument to the <HEAD3> tag, the ampersand must be coded with the <AMPERSAND> tag. In the text, the ampersand can be coded with the ampersand character.

2.3.6 Coding a Hyphen in Text

When two or three hyphens are used consecutively in proportionally-spaced text, VAX DOCUMENT translates them into en dashes and em dashes. (An en dash is slightly longer than a hyphen, and an em dash is slightly longer than an en dash.) This does not occur in monospaced text, such as text in examples. Table 2–2 illustrates how to code a hyphen, an en dash, an em dash, and a minus sign in text and gives an example of each use.

Table 2–2 How to Code a Hyphen in Text

Character Produced	Sample of SDML coding	Result of SDML coding
hyphen	`two-column`	two-column
en dash	`1981--1983`	1981–1983
em dash[1]	`That is -- what I mean to say`	That is—what I mean to say
em dash[1]	`That is---what I mean to say`	That is—what I mean to say
minus sign	`<MATH>(a<MINUS>b)`	$a - b$

[1] An em dash may be produced either by using three consecutive hyphens with no space on either side of them, or by two consecutive hyphens with space on each side of the hyphens. Two consecutive hyphens without spaces surrounding them will produce an en dash.

2.3.7 Coding Tab Characters in Text

VAX DOCUMENT processes tab characters in an SDML file into spaces. For example, if you had pressed the TAB key three consecutive times to insert space into your SDML file, VAX DOCUMENT would interpret the three tabs as three spaces, but would output them differently depending on whether the three spaces occurred in monospaced or proportionally-spaced text.

If those three tabs occurred in a monospaced example (for example, within <CODE_EXAMPLE> and <ENDCODE_EXAMPLE>), they would be output as three spaces; however if they occurred in proportionally-spaced text (for example, within a paragraph), then the three spaces would be replaced by a single space.

Generally, you should avoid using the TAB key in order to eliminate confusion about the spacing in the SDML file; however, entering tabs into an SDML file is not invalid.

3 Using Common Tags

This chapter discusses the coding of text elements frequently used in almost every doctype. These text elements include the following:

- Paragraphs
- Headings
- Chapters
- Front Matter text elements, including title page, copyright page, and preface
- Back Matter elements, including appendixes and glossary
- Lists
- Tables
- Figures
- Examples

This chapter describes some of the most basic tags used in any doctype within VAX DOCUMENT. The tags are also described in Chapter 9.

3.1 Paragraphs

A paragraph is labeled with the <P> tag. Within a doctype, the tag creates the same output, no matter where it is placed in a document. It is unnecessary to leave blank lines before or after the paragraph, or to recall any of the other usual formatting conventions required when a typewriter is used. For example, spacing, justification, and indention of a paragraph are provided by the specified doctype. As writer, you must only inform the text processor that you want this specific text element. Anywhere you place a <P> tag, the text processor will begin a new paragraph. The only reason that you might want to leave blank lines or spaces in your file is to make it easier to read your source file.

The following example shows the use of the <P> tag in a doctype that calls for space between paragraphs, with no indentation.

```
<P>The species is the basic unit of classification.  Clearly,
not all species resemble each other to the same degree or
are closely related.  <P>In any careful study,
one of the vital early steps is the organization and naming of objects.
```

Output

```
The species is the basic unit of classification.  Clearly, not all species
resemble each other to the same degree or are closely related.

In any careful study, one of the vital early steps is the organization
and naming of objects.
```

Note that the paragraphs are formatted correctly no matter how your source file is broken across lines.

If you want a paragraph enclosed in parentheses, be careful to place a space or an end of line between the tag <P> and the parenthetical text. For example:

```
The species is the basic unit of classification.
<p> (Clearly, not all species resemble each other to the same degree or
are closely related.)
```

3.2 Headings

Headings are intended to label a section of text. Depending on the doctype, headings can be numbered automatically during processing. VAX DOCUMENT keeps a count of each heading level and consecutively numbers each heading within a level.

A first level heading is labeled with the tag <HEAD1> and the output is in the form x.x. In the same way, a second level head is labeled with the tag <HEAD2> . You can use up to six heading levels, as required.

The number of a chapter is not counted as a heading level. If the document has a <CHAPTER> tag, the chapter number is included in the heading level number. For example, this chapter is number 3, and so the heading level number of this section became 3.2. If there is no <CHAPTER> tag, only the heading level number is used.

Example

```
<HEAD1>(Physical Features)
<P>Alaska consists of a compact central mass  . . .

<HEAD2>(General Physiographic Features)
<P>The main features are mountains  . . .
<HEAD3>(Physiographic Features)
<P>Between the Pacific and Rocky Mountain systems  . . .
```

This example shows the use of the <HEADx> tag for creating three heading levels. To see the output of this example, refer to the MANUAL doctype chapter in *VAX DOCUMENT Design Samples.*

3.3 Chapters

The <CHAPTER> tag is used to label the chapter title at the beginning of a new chapter. If a source file contains a chapter in a book, the <CHAPTER> tag must be the first tag in the file, as it enables all cross-references within that file. The tag requires a chaper-title as its first argument. It can accept a symbol-name (for the chapter-title) as its second argument and requires one if the file is used in a bookbuild.

A chapter is a basic text element in many types of documentation. However, it is treated as a book element during a bookbuild. You can make chapters of a book number correctly during processing only if the chapter is listed as a book element in the book's profile, and the profile is processed through a bookbuild procedure. An example of the beginning of a chapter follows:

```
<CHAPTER>(Introduction to VAX DOCUMENT)
```

In this example the chapter tag has one argument, the title for that chapter. The first chapter tag in a book automatically receives the chapter number 1, and each following chapter is incremented by one.

If you prefer to maintain a document in a single source file, you can include more than one chapter in a file, and therefore more than one <CHAPTER> tag. The chapters are still numbered automatically. However, you would have to move any chapter manually if you want to reorganize the document.

3.4 Front Matter

The front matter of a book typically consists of four components:

- The title page
- The copyright page
- The table of contents (discussed in Chapter 7)
- The preface

The profile for your book should name a single book element as the front matter. This book element file might contain all the front matter material or it might use the <INCLUDE> tag to bring in separate SDML files containing front matter material, such as the title page, copyright page and preface.

The following is an example of front matter in a single SDML file:

```
<FRONT_MATTER>(front)
<TITLE_PAGE>
<TITLE>(User Manual, Volume 1)
<ORDER_NUMBER>(11-222-333)
<ENDTITLE_PAGE>

<COPYRIGHT_PAGE>
<PRINT_DATE>(August 1988)
<COPYRIGHT_DATE>(1988)
<ENDCOPYRIGHT_PAGE>

<CONTENTS_FILE>

<PREFACE>(11)
<PREFACE_SECTION>(Intended Audience)
.
.
<PREFACE_SECTION>(New and Changed Features)
.
.
<ENDPREFACE>
<ENDFRONT_MATTER>
```

3.4.1 Title Page

The title page is produced by a group of context-sensitive tags. The title page consists of the following tags:

```
<TITLE_PAGE>
  <TITLE>( . . . )
  <ORDER_NUMBER>( . . . )
  <ABSTRACT>
    .
    .
    .
  <ENDABSTRACT>
  <REVISION_INFO>( . . . )

<ENDTITLE_PAGE>
```

The <TITLE_PAGE> tag is defined only within the context of the <FRONT_MATTER> tag. The title page of this book illustrates the output of each of the tags.

3.4.2 Copyright Page

The copyright page is produced by a group of context-sensitive tags. The copyright page tags include:

```
<COPYRIGHT_PAGE>
  <PRINT_DATE>( . . . )
  <COPYRIGHT_DATE>( . . . )
<ENDCOPYRIGHT_PAGE>
```

The <COPYRIGHT_PAGE> tag is defined only within the <FRONT_MATTER> tag. The copyright page of this book illustrates the output of each of the tags.

3.4.3 Preface

The <PREFACE> tag is defined only within the front matter. The preface pages of this book illustrate the type of output these tags produce for this book's doctype and destination.

The preface always begins on a right-hand page following the table of contents, thus it always begins with an odd page number. However, because the table of contents is produced separately from the rest of the book, VAX DOCUMENT cannot automatically supply the correct starting page number for the preface.

When you are doing a final bookbuild, you must determine the actual number of pages in the table of contents and then, if necessary, edit the file that contains the preface, supplying the desired page number as an argument to the <PREFACE> tag.

Calculate the preface page number by adding two to the number of pages in the table of contents, to account for the title and copyright pages. If this number is even, add one to it and use the result as the argument to the <PREFACE> tag. For example, if the table of contents requires six pages, the first part of the calculation yields eight, an even number. Add one and use nine as the argument to the <PREFACE> tag. However, if the sum of two plus the number of pages in the table of contents is odd, add two to the result to obtain the next highest odd number. Then reprocess the front matter with the DOCUMENT command to obtain front matter output that displays the correct page numbers.

3.5 Back Matter

The back matter of a book can consist of the following:

- Appendixes
- Glossary
- Index (discussed in Chapter 7)

3.5.1 Appendixes

Appendixes are structured exactly the same as chapters. They consist of hierarchically numbered sections containing paragraphs, illustrations, and so forth. Use the <APPENDIX> and <ENDAPPENDIX> tags to enclose each appendix. Appendixes are automatically given sequence letters in the order corresponding to the order in which they appear relative to one another in the bookbuild. Thus, you need to give your appendixes symbolic names so that you can refer to them in the book.

3.5.2 Glossary

The glossary is a simple structure that is produced using a template. The glossary template consists of the following pattern of tags:

```
<GLOSSARY>(Glossary\gloss_chap)
   <GTERM>( . . . )  <GDEF>( . . . )
   <GTERM>( . . . )  <GDEF>( . . . )
    .
    .
    .
<ENDGLOSSARY>
```

The glossary begins automatically on a right-hand page with a proper heading. You should arrange glossary terms alphabetically.

3.6 Lists

There are several types of lists:

ALPHABETIC	The list element identifiers are alphabetic letters.
CALLOUT	The list element identifiers are reverse-print callout numbers (on supported output devices), for example ❺.
NUMBERED	The list element identifiers are arabic numerals.
ROMAN	The list element identifiers are roman numerals.
SIMPLE	There are no list element identifiers.
STACKED	List elements do not have identifiers, but are stacked within the specified set of delimiters (braces, brackets, or double brackets).
UNNUMBERED	List element identifiers are special characters, such as bullets.

The following tags are needed to create any type of list:

- <LIST> (with a required argument that specifies the type of list)
- <LE> (meaning "list element")
- <ENDLIST>

The following example shows the use of the <LIST> tag to create an unnumbered list.

Example

```
<P>Mammalian classification of the order Lagomorpha follows:
<LIST>(UNNUMBERED)
<LE>Kingdom Animal
<LE>Phylum Chordata
<LE>Class Mammalia
<LE>Order Lagomorpha
<LE>Family Leporidae
<LE>Genus Lepus
<LE>Species Lepus californicus
<ENDLIST>
```

Output

Mammalian classification of the order Lagomorpha follows:

- Kingdom Animal
- Phylum Chordata
- Class Mammalia
- Order Lagomorpha
- Family Leporidae
- Genus Lepus
- Species Lepus californicus

As shown in this example, a <LIST> tag requires a terminating tag, <ENDLIST>. The <ENDLIST> tag terminates any type of list and does not require an argument.

The <LE> tag identifies each element within the list.

3.7 Tables

The <TABLE> tag is used to create both *informal* and *formal* tables. The two types of tables differ in the following ways:

Informal	Given no caption, is unnumbered and is not listed in the table of contents.
Formal	Given a caption and numbered. Listed in the table of contents. Can be referenced with the <REFERENCE> tag.

Only four tags are required to create either type of table:

- <TABLE>
- <TABLE_SETUP>
- <TABLE_ROW>
- <ENDTABLE>

Table 3–1 describes the four required tags. It also describes several other tags that are not required to create a table but are useful to specify special formatting. Table 3–1 lists the tags in the order that they should be used in your SDML file. Note that the <TABLE_ROW> tag can be used throughout a table. In Table 3–1, arguments are not shown for any tag except for the <TABLE> and <TABLE_ROW_BREAK> tags.

Table 3–1 Table Tags

Tag, in order of use	Description
<TABLE> (table-caption\symbol-name)	Creates a table and enables all other table tags. The optional symbol-name argument can be used to reference the table.[1]
<TABLE_ATTRIBUTES>	Specifies special formatting for the table, such as wide, multipage, or a table that is not to be broken between pages. The table's attributes are ignored if this tag does not precede <TABLE_SETUP> .
<TABLE_SETUP>	Establishes, or "sets up," the number of columns in the table and the table width. If excluded, the table will not be created. You always list each column width except for the width of the last column.
<TABLE_HEADS>	Specifies headings for the table columns.
<TABLE_ROW>	Specifies text for one row of the table. Each argument supplies the text for a column in a row. Can precede or follow table units (grouped sections of the table) as required. Can be repeated throughout the sequence of table tags.
<TABLE_ROW_BREAK> (first) <TABLE_ROW_BREAK> (last)	Specify the boundaries within which a table can be broken onto a new page. Not required, but useful for controlling the format of the table.
<VALID_TABLE_ROW_BREAK>	Controls the pagination of long table rows. Required in tables with long table rows.
<ENDTABLE>	Terminates <TABLE> .

[1]In the <TABLE> tag, the table-caption argument is required in a first level formal table, but is not permitted in a nested table (a table embedded in another table).

A second group of table tags exists to perform special table functions. For example, you can embed a heading in a table that spans several of the table's columns, or you can embed a table within another table. These special tags are not required to create a table, but are available for more complex formatting needs.

Table 3–2 lists the special table functions available, the tags required for each function, and the location where an example of each function can be found in Chapter 9.

Table 3–2 Special Table Functions

Function	Tags Used	Tag Description	Location of Example of Function in Chapter 9
Aligning numbers in a table.	<ALIGN_CHAR>	Specifies a special character that is replaced by a space wherever it appears in table rows. The character can be used to achieve vertical alignment within a column.	See <ALIGN_CHAR>
Nesting (embedding) a table within another table.	All table tags are used. <NESTED_TABLE_BREAK> is required.	Marks a place that a nested table may be broken across pages.	See <NESTED_TABLE_BREAK>
Creating a table key (legend) to explain abbreviations used in the table.	<TABLE_KEY>	Begins a key for the table. If used, it must follow <TABLE_SETUP> .	See <TABLE_KEY>
	<ENDTABLE_KEY>	Terminates the <TABLE_KEY> tag.	
	<TABLE_KEYREF>	Specifies that the table key or legend should be printed below the table. Required if a table key or legend is included in the table.	
Embedding a heading in a table that spans the length of several table columns.	<SPAN>	Causes a heading to span more than one table column.	See <TABLE_UNIT>
	<RULE>	Specifies that a horizontal rule should appear below the embedded heading.	
Grouping sections of a table and labeling the sections.	<TABLE_UNIT>	Divides a table into groups. The groups can have headings within the table.	See <TABLE_UNIT>
	<TABLE_UNIT_HEADS>	Specifies the table unit headings. If used, must follow <TABLE_UNIT> .	
	<ENDTABLE_UNIT>	Terminates the <TABLE_UNIT> tag.	
Including in the source file a separate file that contains a table.	<TABLE_FILE>	Includes a separate file which contains a table. Not required.	See <TABLE_FILE>

Table 3–2 (Cont.) Special Table Functions

Function	Tags Used	Tag Description	Location of Example of Function in Chapter 9
Referring to the symbol-name given to the table.	<REFERENCE>	If a symbol-name is specified in a <TABLE> tag, this tag can be used to reference the symbol name in other places in the document, thereby always producing the correct table number and caption.	See <REFERENCE>

Example

```
<P>See <REFERENCE>(heart_rate_tab) for a list of
the heart rates of selected mammals.

<TABLE>(Heart Rates of Selected Mammals\heart_rate_tab)
<TABLE_ATTRIBUTES>(keep)
<TABLE_SETUP>(3\19\16)
<TABLE_HEADS>(Common Name\Weight\Heart Rate, Beats per Minute)
<TABLE_ROW>(European hedgehog\500-700 g.\246)
<TABLE_ROW>(Gray shrew\3-4 g.\782)
<TABLE_ROW>(Least chipmunk\40 g.\684)
<TABLE_ROW>(Gray squirrel\500-600 g.\390)
<TABLE_ROW>(Harbor porpoise\170 kg.\40-110)
<TABLE_ROW>(Mink\0.7-1.4 kg.\272)
<TABLE_ROW>(Harbor seal\20-25 kg.\18-25)
<ENDTABLE>
```

The output of this example follows:

Output

See Table x–x for a list of the heart rates of selected mammals.

Table x–x Heart Rates of Selected Mammals

Common Name	Weight	Heart Rate, Beats per Minute
European hedgehog	500-700 g.	246
Gray shrew	3-4 g.	782
Least chipmunk	40 g.	684
Gray squirrel	500-600 g.	390
Harbor porpoise	170 kg.	40-110
Mink	0.7-1.4 kg.	272
Harbor seal	20-25 kg.	18-25

3.7.1 Controlling Table Attributes

You can control the following attributes of tables:

- Formality (that is, if the table has a number and a caption or not)
- Its overall width and the size of the text

- Whether it is allowed to break across pages or if the entire table must be kept on a single page

The formality is controlled by the presence of the *caption* argument in the <TABLE> tag. If you do not specify this argument, no number is assigned to the table. Tables are automatically numbered. The numbers are sequential throughout a document, or numbered beginning at 1 in each chapter. With the exception of the REPORT document type, you cannot control the numbering; that is an attribute of the overall document type and design.

You have control over the table attributes of width (the table's margins) and page-breaking. These two attributes are discussed in the following sections.

3.7.1.1 Controlling a Table's Margins

The width of a table, that is the left and right margins, is determined by the following default attributes:

- For formal tables, the left margin of the table is, by default, the same as the normal left margin of text, regardless of whether the text immediately preceding the table is indented.
- For informal tables, the left margin of the table is the same as the current left margin of the text. If the text is indented, (for example if it is a list element) the table is indented at the same margin.
- In either case, the right margin of the table is always the same as the right margin of the text.

For example, if you coded Table x-x without arguments to the <TABLE> tag, and in the context of a list, it would be output as follows:

Output

- These are the heart rates:

Common Name	Weight	Heart Rate, Beats per Minute
European hedgehog	500-700	246
Gray shrew	3-4 g.	782
Least chipmunk	40 g.	684
Harbor porpoise	170 kg.	40-110
Mink	0.7-1.4 kg	272
Harbor seal	20-25 kg.	18-25

If you specify table width values that extend the table beyond the right margin of text, the following actions occur:

- The text formatter makes an attempt to switch the size of the text in the table to a smaller size to see if the table will fit.
- If the table still does not fit, the text formatter issues a warning message; the device converter normally issues a message also. These messages alert you that you must make some adjustments in your table.

In many cases, you can lower the width values to the <TABLE_SETUP> tag to reduce the widths of the individual columns.

In some doctype designs, there is a wide left margin. In those doctypes, the keyword attribute WIDE to the <TABLE_ATTRIBUTES> tag tells the text formatter to use the full page width for the table. Here is the same table coded with <TABLE_ATTRIBUTES> (WIDE):

Common Name	Weight	Heart Rate, Beats per Minute
European hedgehog	500-700	246
Gray shrew	3-4 g.	782
Least chipmunk	40 g.	684
Harbor porpoise	170 kg.	40-110
Mink	0.7-1.4 kg	272
Harbor seal	20-25 kg.	18-25

The <TABLE_ATTRIBUTES> tag, if specified, must precede the <TABLE_SETUP> tag in your SDML file.

You can use the attribute WIDE to avoid having the text formatter switch to a smaller text size. If you require the smaller text size, you can specify <TABLE_ATTRIBUTES> (MAXIMUM).

3.7.1.2 Controlling Page Breaks in Tables

One of VAX DOCUMENT's more powerful features is its ability to handle long tables, that is, tables that require more than a page of output. By default, when the text of a table does not fit on a single page, VAX DOCUMENT assumes that it can break the table across pages, and prints as much of the table as it can on each page.

When a table is continued across several pages, the text formatter repeats the following text at the top of the second page and each subsequent page of output:

- The table number and caption (with the text "Cont'd") if the table is a formal table[1]
- The table column headings, if any

If a table is short and you want to keep it on a single page, you can specify <TABLE_ATTRIBUTES> (KEEP). In this case, the text formatter determines if the table fits on the current page. If the table does not fit, the text formatter starts a new page of output, leaving the page preceding the table short, that is, with blank space at the bottom.

You can control the page breaks in multipage tables in several ways:

- Use both the <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) tags to indicate good first and last places at which page breaks are allowed.
- Group sequences of table rows that must be kept together using the <TABLE_UNIT> tag.

In some cases, you may also need to provide special information to the text formatter to help in breaking complex tables.

[1] The continued caption's placement and the continuation text may vary from one doctype to another.

The following list summarizes the rules for breaking tables. If you are having difficulty getting a table to come out correctly, review these rules to see what you might need to do.

Rules for Page Breaking in Tables

The following list gives some basic rules for page breaking within tables.

- <TABLE_ROW_BREAK> (first) specifies the first place within a table at which a page break is acceptable. You can choose this based on the text in the table rows themselves. For example, if each row in the table has only a single line of text, you would probably place the <TABLE_ROW_BREAK> (first) tag after 3 or more <TABLE_ROW> tags. On the other hand, if each row in the table (or even just the first) has ten or more lines of text, you might want to allow breaking after the first row.
- <TABLE_ROW_BREAK> (last) specifies the last place within a table at which a page break is acceptable.
- Between these two tags, the text formatter assumes that it is okay to break the table between any two table rows. It assumes that all text before the first place and after the last place must be on the same page. If the text before the first place or after the last place is more than a page, the text formatter chooses a place to break the table anyway.
- If you do not specify these tags, the text formatter first tries to keep the entire table on one page. If the table does not fit, it breaks the table anyway, fitting as much as it can on each page.
- By default, all table rows within the bounds of a table unit are places where page breaks must not occur. Therefore, the text formatter will always try to keep all the rows in a table unit on the same page. If the table unit does not fit on the page, it breaks the table anyway, fitting as much as it can on each page. Within the bounds of <TABLE_UNIT> and <ENDTABLE_UNIT> , you can specify <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) tags to indicate the best first and last places, as in the table itself.
- If a table has footnotes, the text formatter does all the page breaking. The <TABLE_ROW_BREAK> tags are ignored.
- By default, the text formatter never breaks a table in the middle of a table row. Consider the following example of a table row:

  ```
  -------------    ----------------
                   ----------------
                  (this text may be
                   a half-page or more
                   in length.)
                   ----------------
                   ----------------
  ```

  Because the text formatter always tries to keep this table row on the same page, you might get short, uneven pages. If you want to allow page breaks within a table row, use the <VALID_TABLE_ROW_BREAK> tag in the middle of the column of the table row:

  ```
  <table_row>(column 1 text \ column 2 text
                              --------------
                              --------------
                              <valid_table_row_break>
                              <p>--------------
                              --------------
              )
  ```

You should generally put the <VALID_TABLE_ROW_BREAK> tag immediately before a <P> , <LIST> , or <LE> tag. The <VALID_TABLE_ROW_BREAK> tag is valid only in two- and three-column tables, and is only valid in the last column of the table.

- The text formatter's rule about never breaking a table in the middle of a table row also applies if a column in a table row contains a nested table. For example:

```
<table_row>(column 1 text \ column 2 text
                             --------------
                             --------------
           <table>
           <table_setup>(2\10)
           <table_row>(one\two
                           <p> this is now text in a nested table.
                                  ...
                            Sometimes this text gets long, too.)
              ...
            <endtable>
)
```

  To specify that it is allowable to break a page within this nested table, you must use the <NESTED_TABLE_BREAK> tag. This tag should be placed just before <P> or <LE> tags, where there is already some vertical space. For example:

```
<table_row>(column 1 text \ column 2 text
                             --------------
                             --------------
           <table>
           <table_setup>(2\10)
           <table_row>(one\two
                           <p> this is now text in a nested table.
                                  ...
                            Sometimes this text gets long, too.
                            <nested_table_break>
                            <p>Here is some more text for this
                            column.
                      )
              ...
            <endtable>
)
```

- The more places you indicate acceptable page breaks in a table, the better the text formatter is able to select the best page breaks and not cause a lot of short, uneven pages.
- If you specify KEEP when you code a table tag, the text formatter tries to keep the table all on one page, no matter if it is too long. You receive a warning message that a page is too long, the device converter also issues an error message, and the part of the table that did not fit on the page probably will not print in the output.
- Use the <PAGE> or <FINAL_CLEANUP> (PAGE_BREAK) tags in tables to control paging explicitly. Note, however, that when you control page breaks, you might have unexpected results when you modify the input file for a device other than the one for which you coded the page breaks.

  Whenever the text formatter detects a possible problem, it issues an informational message to alert you to check your output.

- Use the <TABLE_ATTRIBUTES> (CONTROLLED) tag to indicate that you are going to explicitly specify the range in which the table will be allowed to break. You then must use <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) to indicate the first and last allowable page break points. Between these two tags, the table may be broken between any two <TABLE_ROW> tags.
- Use the <SET_TABLE_ROW_BREAK_DEFAULT> tag to provide a default number of rows that must be on the first page of the table. For example, if you specify <SET_TABLE_ROW_BREAK_DEFAULT> (3), any subsequent table will not be broken until after the third row. You might want to use this default if your table consists of single-line items.

3.8 Figures

Use the <FIGURE> tag to indicate a text element that illustrates a point. It might be a graphic file, a hand-drawn illustration that you need to paste in your output file, or a series of keystokes that represents an item in a diagram.

A set of required tags identifies the text as a figure, establishes the figure's attributes, and identifies the end of the figure. A separate set of figure tags can be used to identify whether the figure is line art, an icon, located in a different file that you want included, or simply blank lines that you want inserted so that a figure can be pasted in later.

The tags that enable the context of a figure are <FIGURE> (figure-caption\symbol-name) and <ENDFIGURE> . Within these two tags, you must include at least one tag that labels the figure text element.

Table 3–3 summarizes the figure text element tags available for use in figures.

Table 3–3 Figure Tags

Function	Tags Used	Description of Each Tag, Location of Example in Chapter 9
Including in the SDML file a separate file that contains a figure.	<FIGURE_FILE>	Includes a figure contained in a separate file and 1 pica of space before and after the included figure. See <FIGURE_FILE> .
Leaving blank space for a figure.	<FIGURE_SPACE>	Leaves a user-specified amount of space for a figure to be pasted in later. See <FIGURE_SPACE> .
Including a section of code as a figure.	<CODE_EXAMPLE>	Marks a section of code. See <CODE_EXAMPLE> .
Including a figure that specifies a dialogue between the system and a user.	<INTERACTIVE>	Marks a system and user dialogue. See <INTERACTIVE> .
Identifying a rough sketch produced at the keyboard for draft output.	<LINE_ART>	Identifies the following text as line-art. See <FIGURE> .
Including an illustration of sample text.	<SAMPLE_TEXT>	Distinguishes an illustration of sample text from the surrounding text. See <SAMPLE_TEXT> .
Estimating a good page breaking point in a long figure.	<VALID_BREAK>	Indicates an acceptable page breaking point. Used in the context of <CODE_EXAMPLE> or <INTERACTIVE> . See <VALID_BREAK> .
Referring to the symbol-name given to a figure.	<REFERENCE>	If a symbol-name is specified in a <FIGURE> tag, you can reference the symbol-name in other places in your document via this tag. See <REFERENCE> .

3.8.1 Figure Elements

When placing a figure in your input file, the tags that you use to label the figure as a text element must be used in the context of a <FIGURE> tag. The figure element (for example, a code example or a figure file) is enabled by <FIGURE> . The <FIGURE> tag, itself, does not include the figure element, but assigns the figure's caption and the figure's symbol-name (if used).

Choose the figure element tag based on the type of figure. The valid tags are included in Table 3–3. A figure can consist of more than one type of figure element, and the types of figure elements can be mixed. For example, you can label a small graphic file and some text that accompanies it by using a <FIGURE_FILE> and a <SAMPLE_TEXT> tag:

```
<FIGURE>(How to Load a Tape\tape_fig)
<FIGURE_FILE>(LN03\mydisk:[mydirectory]tape_fig.six\10)
<SAMPLE_TEXT>
<P>This figure illustrates how to load a tape . . .
<ENDFIGURE>
```

The output of this example would be a formal figure titled "How to Load a Tape." The figure in *[mydirectory]tape_fig.six* would be included, along with the sample text explaining the figure.

3.8.2 Controlling Figure Attributes

You can control the following attributes of figures:

- Formality (that is, if the figure has a number and a caption or not)
- The margin at which the caption is printed, if the figure has a caption
- Whether it is allowed to break across pages or if the entire figure must be kept on a single page, or whether it is allowed to "float"

The formality is controlled by the presence of the *caption* argument in the <FIGURE> tag. If you do not specify this argument, no number is assigned to the figure. Figures are automatically numbered. The numbers may be sequential throughout a document, or numbered beginning at 1 in each chapter. With the exception of the REPORT document type, you cannot control the numbering; that is an attribute of the overall document type and design.

The attributes that you have more control over are the margins and the page-breaking. These attributes are discussed in the following sections.

3.8.2.1 Controlling a Figure's Margins

The width, that is the left and right margins of a figure, is determined by the following default attributes:

- For formal figures, the left margin of the figure caption is the same as the normal left margin of text, regardless of whether the text immediately preceding the figure is indented.
- For informal figures, the left margin of the figure caption is the same as the current left margin of the text. If the text is indented (for example, if it is a list element) the figure is indented at the same margin.

The body of a figure, that is, the set of tags that specify the content of the figure, is processed independently of the <FIGURE> tag and its caption. For example, some writers use the <FIGURE> tag to formalize examples so that they can refer to them from anywhere in a document. For example:

```
<FIGURE>(Status Message\stat_fig)
<FIGURE_ATTRIBUTES>(WIDE)
<CODE_EXAMPLE>(WIDE)
%SYSTEM-S-SUCCESS, YOU HAVE SUCCESSFULLY COMPLETED ALL THE EXERCISES
<ENDCODE_EXAMPLE>
<ENDFIGURE>
```

Here, the WIDE attribute is specified in both the <FIGURE> tag and in the <CODE_EXAMPLE> tag to ensure that the caption will align with the example text. You must use the WIDE attribute in the context of the figure's content as well as in the <FIGURE_ATTRIBUTES> tag.

3.8.2.2 Controlling Page Breaks in Figures

The body of a <FIGURE> may consist of one or more individual elements; for instance, two or more graphics figures may comprise a single figure, or monospaced examples may be mixed with graphics in a figure with a single label. Or, a figure may consist of a monospaced example that is more than a page in length. In these cases, you should be aware of the rules for controlling page breaks in figures.

By default, the text formatter makes the following assumptions about a figure:

- It is less than a single page in length
- If it is a formal figure, it is allowed to "float," that is, if it does not fit immediately following its preceding text on a page, the text formatter moves the figure (usually to the top of the next page of output), and fills it in with text to avoid making a short page
- If it is an informal figure, it is not allowed to float

You can use the attribute keywords KEEP and MULTIPAGE in figures as follows:

- KEEP tells the text formatter that you want the figure to be kept with the preceding text and not to float, regardless of whether a page is short
- MULTIPAGE tells the text formatter that the figure has several elements and that page breaks are legal between the elements

If a figure that requires more than a page is coded without the MULTIPAGE attribute, the text formatter issues a message and breaks the figure anyway, according to the following rules:

- If the figure consists of multiple elements such as <FIGURE_SPACE>, <FIGURE_FILE>, and so on, the text formatter will only cause a page break between the individual elements.
- If a figure consists of a monospaced example that is more than a page in length, the text formatter chooses page breaks based on the following:
  - If the example contains <VALID_BREAK> tags, the text formatter uses their positions as valid page break points
  - If the example contains no <VALID_BREAK> tags, but has blank lines, the text formatter chooses the blank lines as valid page break points
  - If the example contains neither <VALID_BREAK> tags nor blank lines, the text formatter puts as much text as possible on each page and breaks the page at the bottom
- Use the <PAGE> or <FINAL_CLEANUP> (PAGE_BREAK) tags in figures to control the page breaking explicitly. Note, however, that when you control the page breaks, you may have unexpected results when you modify the input file or process the input file for a device other than the one for which you coded the page breaks.

3.8.3 Including Graphics Files

Table 3–4 describes the tags that are used for including graphics in source files.

Table 3–4 Tags for Including Graphics Files

Function	Tags Used to Do Function	Description of Each Tag, Location of Example in Chapter 9
Enabling a small graphics file to be placed to the right or the left of your text.	<ICON>	This feature has several restrictions. Refer to the restrictions in the tag description. See <ICON> .
	<ICON_FILE>	Identifies the graphics file to be included in your text.This tag must be used in the context of the <ICON> and <ENDICON> tags.
	<ICON_TEXT>	Accompanies an icon file with text. Must be used in the context of the <ICON> and <ENDICON> tags.
	<ENDICON>	Terminates the <ICON> tag.
Including in the source file a separate file that contains a figure.	<FIGURE_FILE>	Includes a separate figure file. Valid only within the context of <FIGURE> and <ENDFIGURE> . See <FIGURE_FILE> .

An example of how to include a figure follows:

Example

```
<FIGURE>(Keypad Illustration)
<FIGURE_ATTRIBUTES>(Float)
<FIGURE_FILE>(ln03\mydisk:[mydirectory]keypad.six\22)

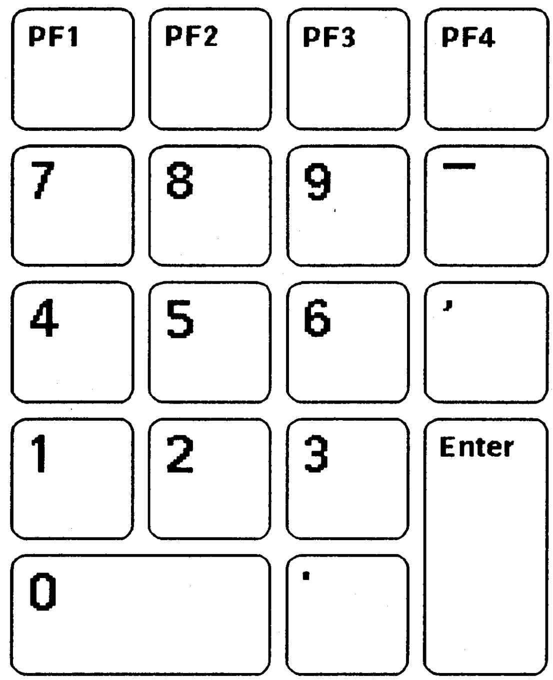
<FIGURE_FILE>(PS\mydisk:[mydirectory]keypad.ps\22)

<ENDFIGURE>
```

This example contains a figure file that includes another file (a graphics file) in this location. The tag requires the graphics file's specification. Chapter 9 contains full descriptions of all the figure tags, including <FIGURE_FILE> .

Output

Figure x–x Keypad Illustration



ZK-7715-HC

3.9 Examples

An example is used to display an excerpt of a programming or command language, the interaction between a user and the system, and the like. There are two types of examples in VAX DOCUMENT, informal and formal. Both types require their own set of tags.

The two types of examples differ in the following ways:

Informal	Remains unnumbered and is not listed in the table of contents.
Formal	Given a caption and numbered. Listed in the table of contents. Can be cross-referenced with the <REFERENCE> tag.

3.9.1 Informal Examples

To create an informal example, use the <CODE_EXAMPLE> and <ENDCODE_EXAMPLE> tags to label your text or code excerpt. An informal example follows:

Example

```
<CODE_EXAMPLE>
$!
$!   The next two lines redefine the cmd. line prompt to be the node
$!   name's initial.
$!
$ NODE :=='F$LOGICAL("SYS$NODE")'
$ SET PROMPT="''F$EXTRACT(1,1,NODE)':"
$!
<ENDCODE_EXAMPLE>
```

Output

```
$!
$!   The next two lines redefine the cmd. line prompt to be the node
$!   name's initial.
$!
$ NODE :=='F$LOGICAL("SYS$NODE")'
$ SET PROMPT="''F$EXTRACT(1,1,NODE)':"
$!
```

3.9.2 Formal Examples

To create a formal example, use the <EXAMPLE> (example-caption\symbol-name) and <ENDEXAMPLE> tags to label your text.

The example shown in the previous section could be made into a formal example as follows:

Example

```
<EXAMPLE>(Login Commands\login_exam)
<CODE_EXAMPLE>
$!
$!   The next two lines redefine the cmd. line prompt to be the node
$!   name's initial.
$!
$ NODE :=='F$LOGICAL("SYS$NODE")'
$ SET PROMPT="''F$EXTRACT(1,1,NODE)':"
$!
<ENDCODE_EXAMPLE>
<ENDEXAMPLE>
```

Output

Example x–x Login Commands

```
$!
$!   The next two lines redefine the cmd. line prompt to be the node
$!   name's initial.
$!
$ NODE :=='F$LOGICAL("SYS$NODE")'
$ SET PROMPT="''F$EXTRACT(1,1,NODE)':"
$!
```

For the purposes of the text formatter, the <FIGURE> tags and the <EXAMPLE> tags are similar in function. The example and figure tags provide an alternate way to label (for cross-referencing purposes) a distinct type of example. See Section 3.8.2 for an explanation of how attributes are processed and how to control page breaks in multipage examples (the information in that section applies to both figures and examples).

In addition to the tags listed for a formal example, you can use any of a second group of example tags inside a formal example to do a specialized function. Table 3–5 lists possible special functions and the associated tags, as well as the location of examples of the functions in Chapter 9.

Table 3–5 Special Example Functions

Function	Tags	Description of Tags, Location of Example In Chapter 9
Beginning an example of interactive dialog.	<INTERACTIVE>	Enables the <S> and <U> tags to distinguish system text from user text. See <INTERACTIVE> and <U> .
	<S>	Labels the system text of an interactive example. See <INTERACTIVE> .
	<U>	Labels the user text of an interactive example. See <INTERACTIVE> .
Specifying a wide format for the example.	<EXAMPLE_ ATTRIBUTES> (wide)	Formats the example's caption as wide. See <EXAMPLE> .
Estimating a good page break point in a long example.	<VALID BREAK>	Indicates an acceptable page breaking point. See <VALID_BREAK> .
Specifying exact placement of the example on the page.	<EXAMPLE_ ATTRIBUTES> (KEEP)	Specifies special formatting for the example. See <EXAMPLE> .
Leaving blank space for an example.	<EXAMPLE_SPACE>	Leaves space for an example to be pasted up during production. See <EXAMPLE_ SPACE> .
Including a separate example file to be placed in your source file.	<EXAMPLE_FILE>	Causes the entire contents of the specified file to be included in your source file. See <EXAMPLE_FILE> .
Referring to a symbol-name given to an example.	<REFERENCE>	If a symbol-name is specified in an <EXAMPLE> tag, <REFERENCE> can be used to reference the symbol-name in other places in the document, thereby always producing the correct example number. See <REFERENCE> .

4 Processing and Printing Files and Books

Chapter 2 and Chapter 3 describe the basics of using the VAX DOCUMENT system, including how to create an input file and how to code the basic text elements of an input file. After you correctly code input files, the next step is to process the files and print or read them on various supported devices.

Processing and printing are done by using the DOCUMENT command line. The components of the command line are discussed in Appendix A. This chapter describes the specific use of these components for processing an individual file or a group of files, and identifies several command line qualifiers used to enhance the process.

Appendix A describes the processing commands and qualifiers that are available from the VAX DOCUMENT command line.

4.1 The Four Types of Processes

The DOCUMENT command is used to process your input files in any of four ways:

- As an individual file (processed in a file build)

  You have an individual file that contains all the SDML tags required to create an entire document. For example, a file containing a letter or a set of overhead slides normally contains all the tags for the complete document. An individual file can contain several chapters, and when processed, the chapters are automatically numbered in the order that you place them in the file. Processing individual files is discussed in Section 4.3.

- As a book (processed in a bookbuild)

  You have a set of SDML files, each of which contains a portion of a book. To do a bookbuild, you must put the text and SDML tags for each individual chapter in a unique file. You then create a file called a profile that lists the files that contain the individual chapters. Bookbuilding is discussed in Section 4.5.

- As an element of a book (processed in a book element build)

  An element is a single file that is a part of a book; for example, the tags and text for a single chapter. When you have made extensive changes to an element of a book, or when a colleague requests a copy of a particular chapter, you can use an element build to process and print the single chapter only. Book element processing is discussed in Section 4.6.

- As a subelement of a book (processed in a book subelement build)

  A book element can be divided into smaller parts, with each part kept in its own file. Each is considered a subelement of the book, and can be processed individually. Subelement processing is discussed in Section 4.7.

Processing a file that is an element or subelement of a book is useful if you need only part of a book for review purposes or if you want to make sure that your references to other parts of the book are correct after you make changes to the input file. You do not have to process the entire book to resolve the references correctly. (Using cross-references is discussed in Chapter 6.)

4.2 Chain of Processing

The chain of processing begins when you invoke VAX DOCUMENT from the command line. Each processor creates an intermediate file and passes that file on to the next processor. The input file is processed in the following order:

1 The tag translator creates a .TEX file.

2 The text formatter creates a .DVI_*device* file.

3 The device converter creates an output specific file, with the file type of LN03, LINE, PS, TERM, or TXT. This final file is sent to the output device for printing or reading through the VMS Mail Utility or on the terminal screen.

No matter the type of build, the input file you specify on the command line is processed in the order of the tag translator, the text formatter, and the device converter, and then sent to a print queue. You can specify that one or more processors be skipped during the processing of a file. The reasons for doing this and the qualifiers used are discussed in the following sections.

4.3 Processing Individual Files

A single generically coded input file can be processed and printed on a hardcopy device, viewed on a terminal screen, or viewed from within electronic mail.

After creating an input file, you process it by typing the DOCUMENT command line using the following syntax:

```
$ DOCUMENT input-file-spec doctype.design destination
```

The command line parameters must be specified in the order shown.

On the command line, the input-file-spec is the name of the input file you want to process. Usually, you do not have to specify the file type of the input file; VAX DOCUMENT assumes the correct file type.

The doctype for your file is the second parameter on the command line. The doctype you choose depends on the type of information you are documenting and on the document design you prefer. Certain doctypes contain tags specific to certain types of information. For more information on these doctype-specific tags, see *VAX DOCUMENT User Manual, Volume 2*. If you are not sure which doctype to use, you can look through *VAX DOCUMENT Design Samples* to choose an appropriate design.

Specify the destination to which you intend to send the output as the third parameter. When processing an individual file, you can specify any of five destinations on the command line: three supported types of printers, a terminal, and an electronic mail destination. The keywords you use to specify these destinations are shown in Table 4–1.

Your system manager may set up site-specific destination keywords for your installation. Destination keywords can be abbreviated.

Table 4–1 Destination Keywords

Keyword	Destination
LINE	Any line printer supported at your site. Produces a file that uses overstriking to achieve bold and underlined text.
LN03	The DIGITAL LN03 or LN03 PLUS laser printers.
MAIL	The VMS Mail Utility. Produces a file that does not display bold or underlined characters.
PS	Any supported POSTSCRIPT device.
TERMINAL	Any ANSI terminal, such as the VT100, VT200 or VT300 series. Produces a file that uses ANSI control sequences to achieve bold and underlined text.

The command line accepts qualifiers to control aspects of the processing. Those qualifiers are discussed in this chapter; the entire set of qualifiers is explained in Appendix A.

After being processed, the file is printed automatically (except when using the terminal or mail destinations) unless you specify the /NOPRINT qualifier on the command line.

Note: **When processing interactively, it is good practice to use the /LIST qualifier on the DOCUMENT command line to capture any error messages. The LIS file that is created to hold the messages is placed in your current directory and named input-file-spec.LIS. To use this qualifier on the command line, type the following command line:**

```
$ DOCUMENT input-file-spec doctype destination /LIST
```

4.4 Controlling File Processing

So far, this chapter has mentioned several qualifiers that, when added to the command line, allow you to process book elements, include other input files into the final output, or map the files processed during a bookbuild. DOCUMENT command line qualifiers also can be used to control how processing occurs, which processors will be used, and what portion of a document gets processed.

Each qualifier should be specified on the command line only once. If a qualifier is specified more than once, only the last use of that qualifier takes effect. The following qualifiers are discussed in this section:

/BATCH	Causes the process to be done in batch mode.
/CONDITION	Sets a condition for the file being processed.
/DEVICE_CONVERTER=(HORIZONTAL_OFFSET=*points*) /DEVICE_CONVERTER=(VERTICAL_OFFSET=*points*)	Each of these qualifiers alters the default position of the text page for the whole document.
/DEVICE_CONVERTER=(STARTING_PAGE=*folio*) /DEVICE_CONVERTER=(ENDING_PAGE=*folio*) /DEVICE_CONVERTER=(NUMBER_OF_PAGES=*total-pages*)	Each of these qualifiers processes and prints only those pages selected.
/INCLUDE	Includes an additional SDML file before the input-file.
/KEEP	Specifies that the intermediate files should be kept in the current directory.
/NODEVICE_CONVERTER	Processes a file without running it through the device converter.
/NOTAG_TRANSLATOR	Processes a file without running it through the tag translator.
/NOTEXT_FORMATTER	Processes a file without running it through the text formatter.
/OUTPUT	Specifies a new name for the output file.
/SYMBOLS	Includes a file containing symbol name definitions for symbols that are referenced in a source file.

4.4.1 Keeping Intermediate Files (Using /KEEP)

By default, the intermediate files are automatically deleted from your directory after they are used in the processing chain. You can keep the intermediate files for later use by specifying the /KEEP qualifier on the command line.

If you keep the intermediate files, you can reprocess a document from any stage of the processing chain. By specifying the appropriate combination of command line qualifiers, processing starts at the specified point:

- To start the processing chain after tag translation, specify /NOTAG_TRANSLATOR on the VAX DOCUMENT command line. You must have a TEX file to start at this point.
- To start the processing chain after text formatting, specify /NOTAG_TRANSLATOR and /NOTEXT_FORMATTER on the VAX DOCUMENT command line to suppress the tag translator and the text formatter. You must have a DVI_*device* file to start at this point.
- To print the output file without reprocessing it, specify /NOTAG_TRANSLATOR, /NOTEXT_FORMATTER, and /NODEVICE_CONVERTER to suppress the tag translator, the text formatter, and the device converter. You must have a device-specific output file, such as a FILENAME.LN03 file.

4.4.2 Processing in Batch Mode (Using /BATCH)

You can specify a qualifier on the command line that submits your DOCUMENT command for processing as a batch job, instead of running it interactively (interactive processing occurs by default). By specifying /BATCH with the rest of your command line, you can continue processing while your terminal is free for other use. Also, by default, the /BATCH qualifier creates a log file, with the same file name as the input file and a file type of .LOG. The log file, which is placed in your current directory, contains a listing of any processing errors found by the tag translator or by the text formatter. The log file can be typed or edited.

When processing in batch, there is no need to use the /LIST qualifier if the batch log file is printed or kept.

The /BATCH qualifier accepts many keywords to modify the batch processing activity. These keywords are the same as the DIGITAL Command Language (DCL) SUBMIT command qualifiers, which are explained in the *VAX/VMS DCL Dictionary*. An example is shown in Appendix A of this manual.

4.4.3 Printing an Existing File

Ordinarily, a file is printed automatically when you issue the DOCUMENT command, unless you specify the /NOPRINT qualifier or the destination as TERMINAL or MAIL on the command line. However, you might want to print a file that has been previously processed and printed or one for which you previously suppressed printing. In this case, specify that the file does not require tag translation, text formatting, or device conversion, by using the following qualifiers on the command line:

```
$ DOCUMENT file-spec doctype destination /NOTAG_TRANSLATOR -
$_/NOTEXT_FORMATTER  /NODEVICE_CONVERTER
```

Print only those files that were successfully processed through the device converter. Use the DOCUMENT command line, specifying the command line qualifiers /NOTAG_TRANSLATOR, /NOTEXT_FORMATTER, and /NODEVICE_CONVERTER.

You can use the DCL PRINT command to print files if you want to get a hard copy of your input file, including the untranslated code, or if you want a hard copy of the LOG or LIS file. You can also use the DCL PRINT command to print, with the appropriate qualifiers, any of your final output files, as shown in Table 4–2. The queue names are defined by your system manager.

Table 4–2 DCL Commands for Printing Files

Type of Output File	DCL Command
LINE, TXT, or TERM	$ PRINT filename /QUEUE=queue-name
LN03	$ PRINT filename /QUEUE=queue-name /NOFEED /PASSALL
PS	$ PRINT filename /QUEUE=queue-name /PARAMETER=(DATA_TYPE=POSTSCRIPT)

4.4.4 Reprocessing a File for a Different Destination

You cannot reprocess a DVI_*device* file to make the file readable by a device other than the one for which the file was originally processed. The exceptions to this rule are files processed for the LINE, TERMINAL, and MAIL destinations. These destinations all use the same DVI_LINE intermediate file, so you can reprocess for these different destinations. Files processed for LN03 or PS destinations should be sent to a different destination by specifying the following command line:

```
$ DOCUMENT file-spec doctype destination
```

Or you can reprocess a TEX file (if the TEX file was kept) by skipping the tag translation stage:

```
$ DOCUMENT file-spec.TEX doctype destination /NOTAG_TRANSLATOR
```

The TEX file has already been processed through tag translation, so the /NOTAG_TRANSLATOR qualifier specifies that the file should not be processed through the tag translator again.

Note: You cannot reprocess a TEX file using a different doctype than the one specified on the original command line. This could cause errors and the results are unpredictable.

4.4.5 Processing or Reprocessing Selected Pages (Using /DEVICE_CONVERTER)

If you want to process and print (or reprocess and print) selected pages of a document, you must have kept the DVI_*device* file. To process selected pages, use the /DEVICE_CONVERTER command line qualifier with the following keywords:

- /DEVICE_CONVERTER=STARTING_PAGE=5, which instructs the printer to begin printing with page 5 of the document.
- /DEVICE_CONVERTER=ENDING_PAGE=3, which instructs the printer to print pages 1, 2, and 3.
- /DEVICE_CONVERTER=(START=5, NUMBER=3), which instructs the printer to print pages 5, 6, and 7.

Alternately, you can use the following qualifier:

- /DEVICE_CONVERTER=(NUMBER_OF_PAGES=*total-pages*) which specifies the number of pages to print, when no ENDING_PAGE keyword is specified.

Note: When you specify more than one keyword to the /DEVICE_CONVERTER qualifier, enclose the keywords in parentheses and separate them by a comma.

The ENDING_PAGE and NUMBER_OF_PAGES keywords cannot be used together on the same command line. If they are used together VAX DOCUMENT will issue an error message.

Refer to Appendix A for more information on these /DEVICE_CONVERTER qualifier keywords.

4.4.6 Altering Page Parameters (Using /DEVICE_CONVERTER)

The /DEVICE_CONVERTER=*(device-keyword)* qualifier contains a useful feature for altering the page positioning of an entire document. The following qualifiers shift the printed image on the page:

- /DEVICE_CONVERTER=(HORIZONTAL_OFFSET= *number-of-points*) is used to specify the text positioning of a document's left margin. The HORIZONTAL_OFFSET keyword shifts the image horizontally.
- /DEVICE_CONVERTER=(VERTICAL_OFFSET=*number-of-points*) is used to specify the text positioning of a document's top margin. The VERTICAL_OFFSET keyword shifts the image vertically.

In both cases, the number of points is an integer that specifies how far from the margin the text should be shifted (72 points is equal to 1 inch). The default horizontal and vertical offsets are both set to 72 points.

4.4.7 Including Additional Files (Using /INCLUDE)

To include an additional file in your final output, specify the /INCLUDE qualifier on the DOCUMENT command line.

The /INCLUDE qualifier specifies that a file is included at the front of your output. It assumes a file type of SDML for the included file if none is specified, but accepts other file types.

An example of a file that might be included with the /INCLUDE qualifier is a cover page that reads "For Internal Use Only." The input file that contains the cover page text, perhaps called COVERPAGE.SDML, would be specified on the command line with /INCLUDE=coverpage.

Another way to include a file is to use the <INCLUDE> (file-spec) tag. This tag serves the same purpose as /INCLUDE, except that the file is included into your output file at the exact location of the tag.

4.4.8 Conditionalizing Files (Using /CONDITION)

VAX DOCUMENT allows you to conditionalize an input file so that you can process the same file to obtain different output for different purposes. The /CONDITION qualifier can be used on the DOCUMENT command line to establish the condition under which the file is processed.

For an example of a conditionalized file, consider the following situation: you want to write two letters, one to a friend and one to an acquaintance. You want most of the text of the two letters to be the same, but you want to add a personal note only to your friend. In this case, you actually need to write only one letter.

If the file that contains the letter is called MYLETTER, you can create conditional sections in MYLETTER, with <CONDITION> (personal) tags, where "personal" is the condition-name. Then, you process this file twice. You process the letter to the acquaintance, using the usual DOCUMENT command line. You also process the letter to your friend, but this time, you use the /CONDITION qualifier on the command line, specifying the condition as "personal." For example:

```
$ DOCUMENT myletter LETTER LN03  /CONDITION=personal
```

When the same condition-name used in the <CONDITION> tag is specified on the command line, all text that was tagged with that condition-name is included in the output. In this example, all of the text tagged with <CONDITION> (Personal) is included. The same text is suppressed when the /CONDITION qualifier is not specified on the command line.

If you have placed <CONDITION> tags in your file, but do not set the condition during processing, no error message is generated. In this case, the conditionalized text is suppressed.

4.4.8.1 Using the <SET_CONDITION> Tag

You can set a condition by using the <SET_CONDITION> tag. Then, you do not use the /CONDITION qualifier when you process the file, but instead set the condition in the file, itself, at the top of the file.

Continuing with the same example, you could place the tag <SET_CONDITION> (personal) in a separate SDML file (called FRIEND_NOTES.SDML) and include that file at the beginning of your input file, MYLETTER.SDML, by using /INCLUDE=FRIEND_NOTES. In FRIEND_NOTES, you could also define symbols to be used only when Condition=personal is set. You can even include text in that file, so that the text is included only when the condition is set.

There is no distinction between a condition that is set with a /CONDITION qualifier on the command line and a condition that is set by a <SET_CONDITION> tag in an input file.

If you set the same condition on both the command line and in the file, no error will result.

4.4.8.2 Setting More than One Condition

It is possible to use more than one condition name to create conditional sections of an input file. You can tag some text elements with one condition name, and other text elements with a different condition name. During processing, the conditional text that does not match a set condition is not included in the output.

To set more than one condition in a file, you must use <SET_CONDITION> tags on all but one condition, because the /CONDITION qualifier can set only one condition.

For information on using conditions during bookbuilding or related processing, see Section 4.6.1.

4.4.9 Assigning a New Output File Name

If you choose to process a conditional file twice to get two different output files, you may want to use a DOCUMENT command line qualifier that assigns a name to the output other than the one assigned by default. /OUTPUT=file-spec gives the final output file any name that you choose.

Continuing with the previous example, the letter to your friend can be processed and the output given a name that clearly identifies it, if you used the following command line:

```
$ DOCUMENT myletter LETTER LN03 /CONDITION=personal /OUTPUT=friend_letter
```

4.5 Bookbuilding

Besides processing an individual file, you can process a group of files as a book. This process is called a *bookbuild.* The major benefit to processing a file as part of a book is that the bookbuild creates a file that contains all the symbol names that you have used throughout the book. You can subsequently reprocess individual elements of the book using the file of symbol names to resolve cross-references to other elements of the book.

There are three steps to processing a group of input files through a bookbuild:

1. Create the book element input files, naming them with SDML file types. These files can be empty but each file must contain a book element tag, and these must have symbol-names. Book elements are discussed in Section 4.5.1.
2. Create a *profile,* naming this file with an SDML extension also. A profile contains a list of <ELEMENT> tags that specify the elements that compose your book. Once created, this profile becomes the file that you specify on the DOCUMENT command line for processing.
3. Process the profile.

Each step in building a book is described in the following sections.

4.5.1 Creating Input Files

When the structure of the book is clearly known, create a file for each element. All of the text need not be written. Use a <CHAPTER> or <APPENDIX> tag to head each element, and add <HEADx> tags if you have an outline of the chapter. The actual paragraphs of text can be filled in when you come to write the chapter.

If you have not yet decided on the order of the chapters, or if the order could change, name each chapter with a descriptive name (for example, INTRO_CHAP.SDML) instead of a numbered title (as in CHAP1.SDML). You can then easily reorder the chapters without renaming the files that contain them.

When creating your book element input files, remember the following points:

- Place each book element in a separate file by placing one of the following book element tags in the file as its first tag:

  <APPENDIX>
  <CHAPTER>
  <FRONT_MATTER>
  <GLOSSARY>
  <PART>

- A book element file should contain only one of the book element tags. For example, two <CHAPTER> tags should not be placed in one file if you want to process that file in a bookbuild. The tag that identifies the file as a book element must be the first tag to appear in the file.
- Each book element tag must contain a symbol-name argument if it will be part of a bookbuild.
- There must be one file for each book element.

4.5.2 Creating a Profile

A profile is an SDML file that contains a list of all of the elements in a book, including the front matter, chapters, appendixes and glossary, if any. The profile also can contain tags that specify where a table of contents or index should be placed in the book and tags that specify logical names for files that are included. Only a specific set of tags can be placed in the profile. All valid profile tags are listed in Table 4–3.

Table 4–3 Profile Tags

Tag	Purpose
<PROFILE>	Signals VAX DOCUMENT that the file is a profile submitted for a bookbuild.
<ELEMENT> (file-spec)	Identifies the file specification of an element of the book.
<INCLUDES_FILE> (logical-name\file-spec)	Defines a logical name for the file specification of a file included within the previous book element. See Section 4.5.3.2.
<CONTENTS_FILE> †	Specifies the location in the book where a table of contents file should be included.
<INDEX_FILE> †	Specifies the location in the book where an index file should be included.
<COMMENT> †	Used to make comments about the elements of the profile.
<ENDPROFILE>	Terminates the profile.

†These tags are also valid in files other than profiles.

To create a profile that reflects the structure of the book, list (in a separate input file) all the files that are part of the book, tagging each file name in the list with an <ELEMENT> tag. In the front of the file, place the <PROFILE> tag, and at the end of the file, place the <ENDPROFILE> tag. Give this SDML file a unique name that indicates it is a profile (for example, MYBOOK_PRO.SDML); processing a profile file that is named the same as one of your other SDML files can cause a fatal processing error.

Each <ELEMENT> tag should specify a file that contains a book element. An example of a profile for a book named *How to Use a Computer* follows:

```
<PROFILE>       <COMMENT>(***Profile for How to Use a Computer***)

<CONTENTS_FILE> <COMMENT>(***insert table of contents here***)
<ELEMENT>(Mydisk:[Mydirectory]intro_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]applications_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]tools_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]conclusion_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]questions_app.sdml)

<INDEX_FILE> <COMMENT>(***insert index here***)
<ENDPROFILE>
```

Each of the book element files listed in the example must contain a book element tag and a symbol name for that element. For example, the chapter TOOLS_CHAP.SDML should begin with the following tag:

```
<CHAPTER>(Using Basic Computer Tools\comp_tools_chap)
```

"Using Basic Computer Tools" is the chapter title and "comp_tools_chap" is the symbol name of the chapter.

4.5.3 Processing a Profile

To perform a bookbuild, use the following command on the command line:

```
$ DOCUMENT profile-spec doctype destination
```

Profile-spec specifies the name of the SDML file containing the profile of the book being processed.

VAX DOCUMENT recognizes that it is building a book when it reads the <PROFILE> tag in the profile. Processing the profile creates a cross-reference (XREF) file for the book. For example, if MYBOOK_PRO.SDML is the profile name, typing the following line creates a complete cross-reference file of all the symbol-names in the book. The cross-reference file is called MYBOOK_PRO.XREF and the final output file (processed for the REPORT doctype) is called MYBOOK_PRO.LN03.

```
$ DOCUMENT MYBOOK_PRO REPORT LN03 /CONTENTS /INDEX
```

You might want to process a profile with the /NOPRINT qualifier so that the book is not printed by default. You must specify /CONTENTS and /INDEX if you want table of contents and index files.

4.5.3.1 Listing the Input Files

A command line qualifier that is useful during a bookbuild is /MAP. This qualifier lists all the input files processed by VAX DOCUMENT in one separate MAP_LIS file, thereby keeping track of the files included and their order. The list starts with the first input file processed and includes any input files specified by <ELEMENT> or <INCLUDE> tags. In the list, files that are included by other files are indented under those files.

If you do not specify a file type, the MAP file is given the same name as the profile with a file type of MAP_LIS.

The /MAP qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /MAP qualifier, the /MAP qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

4.5.3.2 Defining Logical Names for Included Files

To define logical names for included files in book elements, use the <INCLUDES_FILE> tag in your profile. The <INCLUDES_FILE> equates a logical name with a file specification during VAX DOCUMENT execution. The tag's format follows:

```
<INCLUDES_FILE>(logical-name\file-spec)
```

Place the <INCLUDES_FILE> tag in your profile to define a logical name for a file whose contents are included in one of your element files. The first argument to <INCLUDES_FILE> is the logical name for the included file. You then use this name as the argument to the <INCLUDE> tag in one of the book element input files. The second argument specifies the actual file specification into which the logical name translates.

The <INCLUDES_FILE> tag is valid only within a profile. You should place it directly after the <ELEMENT> tag that names the book element file that includes the input file. During a bookbuild, VAX DOCUMENT establishes the logical name definition for each <INCLUDES_FILE> tag before it actually reads and processes the element file name in the preceding <ELEMENT> tag. The logical name remains defined during the processing of later elements.

When an element or subelement is processed by itself (when you use the /PROFILE qualifier on the command line), VAX DOCUMENT again establishes the logical name definitions that were specified by the <INCLUDES_FILE> tags in the profile.

The following example illustrates the use of the <INCLUDES_FILE> tag in a profile:

A writer is drafting a book titled *Purebred Dogs*. The book contains two chapters, one discussing different breeds, and one on the care of purebreds. In addition, there is front matter (title page and preface) and an appendix.

The file identified in each <ELEMENT> tag corresponds to the exact file-name of each book element file. The profile, named CANINE_PRO.SDML, looks like this:

```
<PROFILE>
  <ELEMENT>(FRONT.SDML)
  <ELEMENT>(BREEDS.SDML)
  <ELEMENT>(CARE.SDML)
  <ELEMENT>(APPENDIX.SDML)
<ENDPROFILE>
```

If the book element "CARE.SDML" itself includes two figures and each figure is in a separate SDML file, those figure files could be included by logical-names instead of full file specifications. The logical-names could be defined right in the profile, as follows:

```
<PROFILE>
  <ELEMENT>(FRONT.SDML)
  <ELEMENT>(BREEDS.SDML)
  <ELEMENT>(CARE.SDML)
<INCLUDES_FILE>(bassets_fig\mydevice:[mydirectory]dogcare_fig1.sdml)
<INCLUDES_FILE>(beagles_fig\mydevice:[mydirectory]dogcare_fig2.sdml)
  <ELEMENT>(APPENDIX.SDML)
<ENDPROFILE>
```

In CARE.SDML, you might have the figures referenced and included as follows:

```
<P>The following illustration shows all the basset hound breeds of North
America.
<INCLUDE>(bassets_fig)

<P>In comparison, the next figure illustrates the beagle types throughout
North and South America.
<INCLUDE>(beagles_fig)
```

Notice that the <INCLUDES_FILE> tag in the profile does not actually include the figure, but assigns a logical name to the figure file. Refer to the tag description in Chapter 9 for an explanation of the tag's syntax.

4.5.4 Recovering from Errors

If any of your book element input files contain enough tag translator errors to halt the processor, the tag translator will not produce a .TEX file of that element, and the bookbuild will stop after the last element has been processed. To fix this, you should take the following steps:

1. Fix the errors in the book element input files.
2. Reprocess each revised book element file by typing the following command:

   ```
   $ DOCUMENT input-file-spec doctype destination /NOTEXT_FORMATTER
   /PROFILE=profile-spec
   ```

   The *input-file-spec* is the specification of that individual book element file and the *profile-spec* is the specification of the profile.
3. Submit the bookbuild again, this time using the /NOTAG_TRANSLATOR qualifier to avoid sending the entire book through the tag translator for a second time:

   ```
   $ DOCUMENT profile-spec doctype destination /NOTAG_TRANSLATOR
   ```

Chapter 5 discusses troubleshooting of errors in more detail.

4.6 Processing an Element of a Book

As mentioned in Section 4.1, you can process a book element individually by treating it as a stand-alone file and specifying the file name on the DOCUMENT command line. However, any references to symbol names outside of that element will not be resolved.

To process an individual book element and have all its references resolved correctly, you must have previously processed the entire book through a bookbuild to create an XREF file. Then, to process only one element, you can specify the name of the XREF file to the command line, telling VAX DOCUMENT where to find the symbol-names to resolve the references in the element. The name of the XREF file is specified on the command line with the /PROFILE qualifier.

The XREF file has the same name as your profile, but has the file type of XREF.

To process a book element individually, use the following command on the command line:

```
$ DOCUMENT input-file-spec doctype destination /PROFILE=profile-spec
```

Input-file-spec specifies the input file to be processed. *Profile-spec* identifies the name of the cross-reference file that contains the symbol names of the book.

This command line processes an element of a book and sends it to the specified destination. All symbol-name references within the element are resolved; however, the page numbering begins with 1.

Notice that the /PROFILE qualifier is used only to identify the XREF file during an element or subelement build. You should not use the /PROFILE qualifier when doing a bookbuild because you specify the profile name as the file to be processed.

For more information about referencing symbol names, see Chapter 6.

4.6.1 Building a Conditionalized Book Element

When you build a book element after building the whole book with a condition set, the condition is automatically set in the element build. This happens because the /CONDITION qualifier is stored in the cross-reference file during the bookbuild and then re-used.

If you want to build an element of a conditional book without the condition, you need to specifically override the condition during the element build. At the time of the element build, use the /CONDITION qualifier on the command line, specifying a new condition name. The new condition name supplants the original condition for that element. The new condition name can be any fictitious name (for example, /CONDITION=FAKE). Since you have no <CONDITION> (fake) tags in your book element input file, no text will be eliminated from the output file.

4.7 Processing a Subelement of a Book

If a book element is long enough to be divided into more than one file, creating subelement files, you can process any of those subelements individually. To do this, use the following command on the command line:

```
$ DOCUMENT input-file-spec doctype destination -
 _$ /ELEMENT=file-spec /PROFILE=profile-spec
```

input-file-spec specifies the subelement input file to be processed.

/ELEMENT=file-spec identifies one of the files listed in the profile; the file specified in the input-file-spec is a subelement of this element.

/PROFILE=profile-spec identifies the XREF file containing the book's symbol names.

This command line processes a subelement of a book and sends it to the specified destination. Cross-references within the subelement are resolved and a new XREF file is created; however, the page numbering begins with 1.

Note: **For a subelement build, the book element file and all the other subelement files need to be present, because the tag translator reads the whole element and all its included files even though the entire element is not included in the output.**

You only need to do a subelement build if one of the following conditions exists:

- The file is included in another file that is listed as an <ELEMENT> in a profile.
- The file contains references to symbols defined in other files.
- You want the output file to have the same heading level numbers, table and figure numbers, and so on, as if the file were processed with the entire element.

Heading level numbers, figure numbers, and table numbers, are correct only if they have symbol-name arguments.

You do not need to use a subelement build if you are only processing a file to check text content or spelling. in this case, you can process the subelement file as a stand-alone file.

4.8 Simultaneous Element Builds and Bookbuilds

If two or more users are working on the same book and try to process separate elements of the book simultaneously, specifying the /PROFILE qualifier to reference the XREF file for the book, the XREF file may not be accessible by one of the users and cross-references will not be resolved.

For example, one user opens the XREF file and loads it into memory prior to doing the build on a book element. The tag translator locks the existing XREF file, so that no one else can use it until a new XREF has been successfully written. The file is locked to ensure that any symbol name changes introduced by an element build will get incorporated in subsequent builds.

VAX DOCUMENT protects against the use of an old XREF file by locking the file. A second user will be unable to open the XREF file. That user receives an error message about the XREF file failing to open, and the tag translator halts.

Writers can work independently on separate chapters of a book and can do their element builds without concern for the actions of other writers. But they should be prepared for an error message from the tag translator if they attempt to do an element build at the precise moment that another writer has the XREF file locked.

If you get such a message, wait a minute or two and try the element build again. The lock happens only during the first pass of the tag translator, when it is reconstructing the XREF file, so even though it may take several minutes to completely process a chapter, the actual time that the XREF is locked will be less.

Likewise, when someone starts a bookbuild, and the <PROFILE> tag is encountered, any existing XREF file is locked. Therefore, other users cannot be doing element builds using this XREF file. If the writer doing an element build opens the XREF file right before a bookbuild begins, the bookbuild itself will abort.

A writer doing a bookbuild should coordinate with other writers working on the book. The locking action acts as a safeguard against a book or element build using an old XREF file for resolving cross-references.

5 Troubleshooting SDML Files

During processing, VAX DOCUMENT diagnoses the source file and checks for errors. Error messages are produced when VAX DOCUMENT encounters the following error conditions:

- Misspelled SDML tags and argument names
- Tags used out of proper context
- Missing terminating tags

VAX DOCUMENT issues a message that provides the line number, the probable cause, and other information about the error. The user returns to the source file to correct the error and reprocess the file.

However, in some cases, the file processes without producing any error messages, but the printed output clearly indicates that something has been forgotten or that something needs to be adjusted. A few problems of this nature are quite common and examples of incorrect and correct input are shown in this chapter.

If you have a batch process running for an abnormally long amount of time, have your system manager check that the system parameters are set correctly, including the required space quota amount.

5.1 Error Messages

Error messages and informational messages are produced by each of the processing programs in the VAX DOCUMENT system:

- The tag translator
- The text formatter
- The device converter

VAX DOCUMENT messages have the following general format, consisting of four fields:

```
%FACILITY-S-IDENT, text...
```

Facility

The facility field in the message identifies the program that issues the message.

The following are VAX DOCUMENT facility prefixes:

- %DOC
- %TAG
- %DVC

Severity

The severity field indicates the seriousness of the error.

- I—The message is informational; processing is not affected.
- W—The message provides a warning; processing is not terminated. A warning is produced if arguments are supplied where they are not necessary, or if a tag has been misspelled, such as <TABLER> .

  If the tag translator generates more than 30 warnings by the time it finishes its first pass through a file, it stops processing at the end of that pass. You can see the error messages and fix the errors before reprocessing the file.
- E—A major error has occurred; processing continues, but no output may be produced.
- F—The message indicates a fatal problem; processing is terminated.

  Fatal problems, such as missing terminator tags, must be corrected before the file can be processed further.

Identification

The identification field is an abbreviation of the problem found by the processing program.

Text

The text of the message, if present, describes the problem found by the processing program. The following error message is an example of the message format:

```
%TAG-W-GTMAXARGS at tag <P> on line 1 in file
        disk:[directory]filename.SDML
        More than 0 arguments supplied to tag <P>
```

%TAG-W-GTMAXARGS

The facility, severity, and identification of the message. This warning message was issued by the tag translator.

at tag <P> on line 1 in file

The first line of text provides the line number and the name of the tag being evaluated when the message was issued.

disk:[directory]filename.SDML

The second line of text provides the name of the file being read.

More than 0 arguments supplied to tag <P>

The third line identifies the problem.

If VAX DOCUMENT is running interactively, error messages from the tag translator are displayed on the screen and recorded in a LIS file if the /LIST qualifier is used. The LIS file has the same name as the source file, with the LIS file extension. Error messages from the text formatter are written to a LIS file.

If VAX DOCUMENT is running in batch mode, error messages are automatically written to a LOG file.

5.2 Output Problems

Sometimes processing completes without an error, but when the output is printed, it is easy to see that things have somehow gone awry.

This section illustrates some common solutions to problems that a writer must diagnose after seeing the formatted output.

5.2.1 Incorrect Paragraph Spacing

A common mistake when coding a file is to leave out <P> tags. Most often, the tags are left out after major headings, such as chapter titles and headings.

The problem is only apparent in doctypes that place headings at a different margin than paragraphs.

Incorrect Input

```
<CHAPTER>(symbol-name)
Paragraph text...

<HEAD1>(First-level Heading)
The information in this chapter covers...

<P>Also including is...

<HEAD2>(Second-level Heading)
First item under discussion.

<P>Details...
```

Correct Input

```
<CHAPTER>(symbol-name)

<P>Paragraph text...

<HEAD1>(First-level Heading)

<P>The information in this chapter covers...

<P>Also including is...

<HEAD2>(Second-level Heading)

<P>First item under discussion.

<P>Details...
```

5.2.2 Problems with Examples

Code examples, interactive examples, and syntax examples appear on the printed page as monospaced output. Because of the change to a monospaced font, examples restrict the use of many tags. Tags that require special text formatting, such as lists, tables, and headings, cannot be used within monospaced examples, nor may index tags be used.

When the <CODE_EXAMPLE> tag is used with the <ENDCODE_EXAMPLE> tag, spacing is preserved. However, when the <CODE_EXAMPLE> tag is used with an argument and no terminating tag, any leading or trailing spaces are not preserved.

Incorrect Input

```
<TABLE>
<TABLE_SETUP>(2\10)
<TABLE_HEADS>(Variable\Value)
  <TABLE_ROW>(<CODE_EXAMPLE>(ABC)\<CODE_EXAMPLE>(  1))
  <TABLE_ROW>(<CODE_EXAMPLE>(XYZ)\<CODE_EXAMPLE>(100))
<ENDTABLE>
```

Correct Input

In this example, the writer was trying to use the spaces to align the values in the table. The <ALIGN_CHAR> tag produces correct alignment:

```
<ALIGN_CHAR>(#)
<TABLE>
<TABLE_SETUP>(2\10)
<TABLE_HEADS>(Variable\Value)
  <TABLE_ROW>(<CODE_EXAMPLE>(ABC)\<CODE_EXAMPLE>(##1))
  <TABLE_ROW>(<CODE_EXAMPLE>(XYZ)\<CODE_EXAMPLE>(100))
<ENDTABLE>
<ENDALIGN_CHAR>
```

5.2.3 Incorrect Sequencing of Formal Elements

Formal elements can be output out of sequence if coded inconsistently. For example, it is possible to code three figures that the tag translator automatically numbers 1, 2, and 3. In the output, the figures could possibly appear out of sequence, for example, as 2, 1 and 3.

The problem arises if the figures are coded with the FLOAT attribute. After the tag translator numbers the figures, it is possible for the text formatter to float them out of order.

Incorrect Input

```
<FIGURE>(Caption\symbol1)
  <FIGURE_ATTRIBUTES>(FLOAT)
  <FIGURE_SPACE>(10)
<ENDFIGURE>

<FIGURE>(Caption\symbol2)
  <FIGURE_ATTRIBUTES>(FLOAT)
  <FIGURE_SPACE>(10)
<ENDFIGURE>

<FIGURE>(Caption\symbol3)
  <FIGURE_ATTRIBUTES>(FLOAT)
  <FIGURE_SPACE>(10)
<ENDFIGURE>
```

If all three figures are short, any of the figures could float out of order. To correct this problem, code all three figures with the KEEP attribute.

Correct Input

```
<FIGURE>(Caption\symbol1)
  <FIGURE_ATTRIBUTES>(KEEP)
   .
   .
   .
<FIGURE>(Caption\symbol2)
  <FIGURE_ATTRIBUTES>(KEEP)
   .
   .
   .
<FIGURE>(Caption\symbol3)
  <FIGURE_ATTRIBUTES>(KEEP)
   .
   .
   .
```

6 Referencing Symbol-Names

VAX DOCUMENT allows you to create and reference *symbol-names* for any type of information that is subject to change. This ensures that documents are always up to date, especially in regard to the following references:

- References to text elements that might change, such as the name of a product or project.
- References to book elements that have numbers associated with them, such as chapters.

This chapter describes how to create symbol-names for text and book elements and how to reference them.

A symbol-name is a term that you assign to some text which, once created, becomes an easy and reliable way to refer to past or future chapters, tables in other sections of the book, figures, other writers' books, and numerous other references.

6.1 Creating Symbol-Names

You can create symbol-names for both text elements and book elements. Each symbol-name is assigned as a tag argument. For example, to assign a symbol-name to an example captioned *VAXcluster Multi-file Summary*, you could put the following tag into your SDML file:

```
<EXAMPLE>(VAXcluster Multi-file Summary\multi_file_exam)
```

The symbol-name *multi_file_exam* could then be referenced anywhere in the document, with the <REFERENCE> tag:

```
<REFERENCE>(multi_file_exam)
```

When VAX DOCUMENT processes the <REFERENCE> tag, it substitutes the actual name for the symbol-name in your output. In this case the symbol-name "multi_file_exam" would be replaced with "VAXcluster Multi-file Summary."

Symbol-names must not exceed 31 characters, and can consist only of alphabetic letters, numbers, and underscores. Do not begin a symbol-name with an underscore.

The following naming convention might be useful: in each symbol-name, suffix the name with the type of text element for which you are creating a symbol. For example, a symbol-name for a table might be called "american_kings_tab," or for a chapter, "american_kings_chap." This convention makes it easy to identify different types of symbol-names as you edit a source file.

6.2 Storing Symbol-Names in a Cross-Reference File

An automatic function of VAX DOCUMENT is to create a *cross-reference file,* which is a file containing a list of the symbol-names used in one or more files. An entry in the cross-reference file may have some or all of the following information associated with it:

- The *symbol-name.* As previously described, symbol-names must be no more than 31 characters in length, and must have only alphabetic letters, numbers, or underscores in them.
- A *symbol-type.* Each cross-reference file entry must have a symbol-type, which not only indicates the type of element to which a symbol refers (for example, Figure, Table, or Chapter), but may also control the output when the symbol is referenced. For example, if the element is a Chapter type, the output associated with the reference to it is the default text "Chapter."
- A *symbol-value.* Cross-reference file entries for text elements that have numeric values associated with them (header levels, figures, numbered tables, and so on) always associate this number with the *symbol-value.*
- The *symbol-text.* The text associated with a cross-reference file entry is the text string associated with it: the chapter title, a figure caption, heading level text, or a user-defined string.

When you process an individual file that contains symbol-names, those symbol-names are temporarily placed in the cross-reference file (in your computer system's memory). VAX DOCUMENT uses the table to resolve all symbol-name references, then erases the table from memory.

When you process a book through a bookbuild, or an element of a book through an element build, VAX DOCUMENT again creates a cross-reference file and uses it to resolve all references. However, the bookbuilding process requires that the symbol-names be stored for later use, so VAX DOCUMENT saves the cross-reference file by copying it into the directory that contains the profile used to perform the bookbuild. This copied file is named with the profile name and is given the filetype of XREF.

Note: The XREF file is not readable; you should never attempt to edit it. Also, do not delete it; if it is deleted accidentally, you must perform a new bookbuild before any subsequent element builds can be done.

6.3 Creating Symbol-Names for Text and Book Elements

Text and book elements can be divided into two groups, sequenced and nonsequenced elements.

A sequenced element is labeled in the output. The number or letter assigned is relative to the other elements of the same type in the rest of the book. For example, a table is assigned the number three only if there are two tables before it in the book element being processed.

The following tags mark sequenced elements and are used to create symbol-names for these elements:

- <APPENDIX>
- <CHAPTER>
- <EXAMPLE>
- <HEADx>
- <FIGURE>
- <PART>
- <TABLE>

A nonsequenced element is any text string or book element that does not need to be numbered. An often used sentence or statement is an example of a nonsequenced text string. A book title is also a nonsequenced text element, as it does not receive a number.

The following tags are used to create symbol-names for nonsequenced text elements:

- <DEFINE_BOOK_NAME>
- <DEFINE_SYMBOL>
- <FRONT_MATTER>
- <GLOSSARY>

See Chapter 9 for full descriptions of these tags.

The <DEFINE-SYMBOL> and <DEFINE_BOOK_NAME> tags accept a symbol-name as their first arguments. The book element tag <FRONT_MATTER> accepts a symbol-name as its only argument. For all other tags that accept symbol-names, assign the symbol-name as the tag's second argument.

Each book element requires a symbol-name if the book is to be processed through a bookbuild; if you use these tags without symbol-name arguments, no entry is added to the cross-reference file during processing and you receive a warning message.

For example, a chapter might have its title and symbol-name identified as follows:

```
<CHAPTER>(Running the Program\ref_chap)
```

The <CHAPTER> tag creates a cross-reference file entry for this chapter. The chapter's symbol-name is saved in the table as *ref_chap*. Other information associated with this symbol is also saved, including the chapter title, Running the Program, and the chapter number (which is assigned automatically). This chapter's symbol-name, which appears in the <CHAPTER> tag argument at the head of the book element, becomes the key to element building, as described in Section 4.6 of Chapter 4.

6.4 Referencing Text and Book Element Symbol-Names

The <REFERENCE> tag is used to reference any kind of symbol-name. The first argument to this tag is always the symbol-name you want to reference. Any symbol-name specified by the <REFERENCE> tag causes the tag translator to search the cross-reference file for the definition of the symbol-name and to replace the reference with the definition. For example:

```
See <REFERENCE>(ref_chap) for step-by-step instructions.
```

This book element reference causes the tag translator to look in the cross-reference file for the symbol-name *ref_chap* and to replace the tag and argument with the default symbol type keyword associated with the chapter (for example, "Chapter") and the chapter number. Therefore, if the preceding example referenced the third chapter of a book, the output would be as follows:

```
See Chapter 3 for step-by-step instructions.
```

In your source file, the reference to a symbol-name can precede or follow the tag that defines the symbol-name. Therefore, the following line:

```
<Define_Symbol>(plan_tab\The 43rd Project Plan)
```

could be placed anywhere in the document, before or after the following line:

```
<p>The series of steps are identified in <reference>(plan_tab).
```

Relative placement of the two tags is unimportant because the tag translator makes two passes over a document during processing, creating the cross-reference file during its first pass and not using the file to resolve the references until the second pass. Therefore, the correct symbol-names are always entered in the file before the references are translated.

6.4.1 Controlling the Output of Your Reference

The <REFERENCE> tag allows you to control the output of a reference through use of its second argument. You can use this argument to specify that you want either a portion of your reference or the entire reference placed in your output.

Optional second arguments to <REFERENCE> include the following:

- VALUE—specifies that, if a symbol-name has a value associated with it, you want only that symbol-value (number or letter) included in the output.
- TEXT—specifies that you want only the text of a symbol included in the output (for example, a chapter title or table caption).
- FULL—specifies that you want both the value and the text included in the output.

Consider the earlier example of the <CHAPTER> tag. VAX DOCUMENT automatically assigns to the book element symbol-name its symbol-type keyword, "Chapter," and its symbol-value, the sequential number of the chapter during processing. By default, both the symbol-type keyword and the symbol-value are used to replace the symbol-name reference. However, if you want the output to include the associated text of the symbol-name with the symbol-type keyword and value, you need to specify \FULL in the

reference. If "Running the Program" is the title of the fourth chapter, and you want that title to be included in your reference, you would use the \FULL argument. For example:

```
<reference>(ref_chap\FULL)
```

Such a reference would automatically output the following text:

```
Chapter 4, Running the Program
```

When you do not specify a second argument, the output defaults according to the symbol-type. Table 6–1 summarizes the default output for each type of text or book element.

Table 6–1 Element Types and Default Output of Symbol-Names

	Second Argument to <REFERENCE>			
Text Element	**(null)**	**VALUE**	**TEXT**	**FULL**
EXAMPLE	"Example" number	number	caption	"Example" number, caption
FIGURE	"Figure" number	number	caption	"Figure" number, caption
HEADx	"Section" number	number	caption	"Section" number, caption
TABLE	"Table" number	number	caption	"Table" number, caption
BOOK TITLE	The title argument specified in <DEFINE_BOOK_NAME> .			
TEXT STRING	The text-string argument specified in <DEFINE_SYMBOL> .			
Book Elements	**(null)**	**VALUE**	**TEXT**	**FULL**
APPENDIX	"Appendix" letter	letter	title	"Appendix" letter, title
CHAPTER	"Chapter" number	number	title	"Chapter" number, title
PART	"Part" number	number	title	"Part" number, title
SECTION	"Section" number	number	title	"Section" number, title
FRONT MATTER	none	none	none	none
GLOSSARY	none	none	none	none

Here is an example of a figure reference with a defaulted second argument:

```
<P><reference>(keyboard_fig) shows the keypad layout.

<FIGURE>(The Default Keys\keyboard_fig)
<FIGURE_SPACE>(20)
<ENDFIGURE>
```

In this example, the figure titled The Default Keys was assigned the symbol-name of *keyboard_fig* in the tag <FIGURE> . The reference to this symbol-name does not contain a second argument, which means the writer accepts the default output. Therefore, if this reference and figure were to occur in a chapter numbered three, and would be the fourth formal figure, the paragraph containing the reference would have the following output:

Figure 3–4 shows the keypad layout.

If the second argument to the <REFERENCE> tag was \FULL, the output would be as follows:

Figure 3–4, The Default Keys shows the keypad layout.

6.4.2 Referencing Symbol-Names in Other Files

For text strings that you reference frequently in separate files of the same book, it is useful to place all of your <DEFINE_SYMBOL> and <DEFINE_BOOK_NAME> tags in one file, in a symbol definitions file. Then, to reference the symbol-names stored there, specify the name of the symbol definition file with the /SYMBOLS qualifier on the DOCUMENT command line when you process the file.

For example, you could create a symbol definitions file named INTRO_BOOK_SYMS.SDML. This file might contain the following symbol-names:

```
<define_symbol>(PROGRAM_NAME\Matrix Maker)
<define_symbol>(PROGRAM_VERSION\3.2)
```

When processing a file that references either of these symbols, you would specify the name of the symbol definitions file on the DOCUMENT command by typing the following line:

```
$ DOCUMENT filename doctype destination /SYMBOLS=INTRO_BOOK_SYMS
```

The /SYMBOLS qualifier assumes a filetype of .SDML by default.

6.5 Creating a Preliminary Profile

Before you write your book, use the following procedure to make sure that your symbol-names will be referenced correctly in the entire book or in individual elements of the book. If you follow this procedure before you begin to write, your symbol-name references will be correct throughout the writing process.

1. When the structure of the book is clearly defined, create a file for each element. All of the text need not be written. Use the <CHAPTER> or <APPENDIX> tags to head each element, and add <HEADx> tags if you have an outline of the chapter. The actual paragraphs of text can be filled in when you come to write the chapter.
2. Create a profile that reflects the structure of the book. To do this, in a separate SDML file list all the files that are part of the book, tagging each file name in the list with an <ELEMENT> tag. In the front of the file, include the <PROFILE> tag, and at the end of the file, add the <ENDPROFILE> tag. Name this SDML file with any name that indicates it is a profile, for example, MYBOOK_PRO.SDML.

   Each <ELEMENT> tag should specify a file that contains a book element. No other files should be included in the profile.
3. Process the profile to create the cross-reference file for the book. Indicate that you want to build the book by specifying the profile as the filename in the DOCUMENT command line. In this case, MYBOOK_PRO.SDML is the name of the profile.

   ```
   $ DOCUMENT MYBOOK_PRO doctype destination
   ```

After completing this procedure, you have a complete cross-reference table of all your book element symbol-names.

6.6 Adding New Symbol-Names

As you write, you will probably add or delete tags that supply symbol-names. To reference new symbol-names and have each reference resolved correctly, you must rebuild the affected book elements as you go. You might want to rebuild any of the following:

- An entire book (through a bookbuild)
- An individual book element (through an element build)
- A portion of a book element (through a subelement build)

See Chapter 4 for the procedures for reprocessing files.

Rebuilding an Entire Book

You need to reprocess the entire book through a bookbuild only when you change the book in either of the following ways:

- Reorder book elements that have incremental values associated with them, for example, reorder the chapters of the book
- Add additional book elements that affect the numbering of other elements, for example, insert a new chapter

Rebuilding an Individual Element of a Book

To rebuild a file that contains one element of a book, use the /PROFILE qualifier on the DOCUMENT command line. The /PROFILE qualifier causes the tag translator to rebuild the cross-reference file with the newest symbol-names for that element.

Rebuilding a Subelement of a Book

Some books contain a large single element. The element might not be subdivided into chapters or appendixes, but instead consists of many pages of a highly structured reference section. Each item described in the reference section could be written as a separate SDML file, and it becomes convenient to process these files individually. In essence, you want to process a subelement of a book.

To process a subelement, two qualifiers are added to the command line, /PROFILE and /ELEMENT. The tag translator processes the entire element through its first pass to correctly update the cross-reference file, but produces output only for the subelement that you specify as the input file on the DOCUMENT command line.

7 Generating a Table of Contents, Index, and Master Index

This chapter describes how to create a table of contents, an index and a master index for your document. Complete reference information on the tags described in this chapter can be found in Chapter 9.

7.1 Creating a Table of Contents

You create a table of contents by specifying the /CONTENTS qualifer when you process your SDML file. VAX DOCUMENT automatically generates a table of contents entry for each of the following in your SDML file:

- Appendixes, chapters, glossaries, indexes, parts, and prefaces
- Formal examples, figures, and tables (if specified with a caption or a symbol-name argument)
- Numbered heading tags (table of contents entries for the <HEAD5> and <HEAD6> tags are doctype-dependent)
- SOFTWARE doctype reference-element tags: <COMMAND>, <ROUTINE>, <SDML_TAG>, <STATEMENT>, and <FUNCTION>
- SOFTWARE doctype subcommand tags: <SUBCOMMAND> and <SUBCOMMAND_SECTION_HEAD>
- REPORT doctype <SECTION> tag

See the table of contents in this manual for a sample of a table of contents created by VAX DOCUMENT.

The table of contents is created during the text formatting of your document. If you omit text formatting by using the /NOTEXT_FORMATTER qualifier, contents generation is also omitted. If you use the /CONTENTS and /NOTEXT_FORMATTER qualifiers together, VAX DOCUMENT issues an error message.

VAX DOCUMENT creates a separate file for the table of contents. It names this file by taking the file name of the *input-file-spec* parameter you specify on the DOCUMENT command line and appending "_CONTENTS" to it. You need do nothing more than specify /CONTENTS on the command line if you want to maintain your table of contents in a separate file; however, if you want this file automatically incorporated into your document, use the <CONTENTS_FILE> tag in your SDML file.

When the <CONTENTS_FILE> tag is processed by VAX DOCUMENT, the latest table of contents file is incorporated into the final output file where the <CONTENTS_FILE> tag occurs in the SDML file. If you use the /NODEVICE_CONVERTER qualifier, the table of contents will not be incorporated into your file.

You can make sure that the latest version of the table of contents is incorporated into your file by specifying the /CONTENTS qualifier whenever you process a file that contains the <CONTENTS_FILE> tag. This action guarantees that the table of contents file correctly reflects the organization and pagination of your SDML file. If you do not specify /CONTENTS, an out of date table of contents may be included into your document.

When you create a table of contents using the /CONTENTS qualifier, whatever parameters and other qualifiers you specify affect the processing of both the table of contents file and your SDML file. For example, if the LN03 destination and the /NOPRINT qualifier are specified, both files are formatted for the LN03 laser printer (*input-filename*.LN03 and *input-filename*_CONTENTS.LN03) but neither is printed.

The following example shows how a table of contents may be created along with the output of the file MYREPORT.SDML:

```
$ DOCUMENT myreport REPORT LN03 /CONTENTS
```

The table of contents generated by the previous command would be MYREPORT_CONTENTS.LN03.

7.2 Creating an Index

You can automatically generate an index file from your SDML file by using the /INDEX qualifier to the DOCUMENT command. To create an index using VAX DOCUMENT, you mark the text elements within your SDML file that you want included in your index with <X> and <Y> tags, and then you process your SDML file using the /INDEX qualifier to generate an index file from your SDML file.

The <X> tag is used to create main entries in the index. Main entries have page numbers associated with them. The <Y> tag is used to create cross-reference entries in the index. Cross-reference entries (also called "See" or "See also" entries) do not have page numbers associated with them, instead they refer the reader to a main entry in the index.

Both <X> and <Y> tags can have subentries (also called subheadings or modifications), which are index entries that are indented under either a main entry created using the <X> tag, or a cross-reference created using the <Y> tag.

The following example shows a part of an SDML file that contains <X> and <Y> tags. The <XSUBENTRY> tag is used to create subentries in both tags.

```
<head1>(Usage of Official Vehicles)
<X>(Vehicles<XSUBENTRY>usage of)
<Y>(CORP-AUTO report<XSUBENTRY>See Vehicles)
<p>
Official vehicle usage is listed in a separate report CORP-AUTO-1439u2.
This report is organized as in the following outline.
```

If these tags occurred on page 3-1, they would create the following index entries (under the headings "**-C-**" and "**-V-**", respectively):

CORP_AUTO

    See Vehicles

Vehicles

    usage of, 3-1

See the index in this manual for an example of an index produced using VAX DOCUMENT.

The following example shows a command line that will create an index file along with the output of the file MYREPORT.SDML:

```
$ DOCUMENT myreport.sdml REPORT LN03 /INDEX
```

You can tailor your index in several ways by using the arguments and keywords explained later in this chapter, or you can accept the VAX DOCUMENT default index file characteristics. The following list describes the default characteristics of an index file:

- Capitalization of index entries is retained in the index as entered.
- Index entries are formatted in two columns on each page. On the last page the entries are balanced so that the two columns are of equal length.
- Guide headings are automatically inserted into the index before each entry that begins a new alphabetic section. For example, when the first index entry beginning with the letter "B" is encountered, the guide heading "**-B-**" is placed before it. The exact format of the guide heading is doctype-dependent.
- Index entries are sorted in the index according to the following default rules:
  - Entries with all nonalphabetic characters are placed before all other entries.
  - Entries beginning with nonalphabetic characters are sorted by the alphabetic characters following the nonalphabetic characters.
  - Entries are sorted on a word-by-word basis in which spaces and hyphens are treated as significant sorting characters.
  - Index entries generated by the <Y> tag are placed at the beginning of each subentry level. This is done on the assumption that cross-reference information should be positioned at the top of each subentry level so that the user can find it easily.
- Index entries are merged only when they have identical spelling, spacing, emphasis, and capitalization in the SDML file. Page-number references are merged only if they refer to the same page and are specified with identical attributes such as emphasis and appended text. Even if index entries or page numbers look identical on the printed page, they will not be merged unless they are coded identically in the SDML file.

  For example, the entries <X> (<TAG> (LINE)) and <X> (<TAG> (line)) would appear the same on the printed page because the <TAG> tag would format both tag-names in uppercase characters. However, these entries would create separate index entries because the tag-name is not capitalized consistently in the SDML code for both entries.

- Either a bullet (•) or a comma (,) is placed between each index entry and its first page reference. The character used for punctuation is doctype-dependent. For example, if the doctype specifies that commas are to be used, the entry appears as follows:

  Command qualifiers, 4-10

  If a main level index entry is followed by a subentry rather than a page reference, no punctuation is placed after the main entry. Instead, the punctuation is placed after the subentry to set off the page reference as in the following example:

  Command qualifiers

    description, 4-10
    examples, 4-12

7.2.1 Creating Main Index Entries

Use the <X> tag to create a main index entry. The <X> tag places an entry in the index that contains the page number of the output page on which this tag occurs. For example, if the <X> tag in your SDML file is processed while page 2-132 is being created by the text formatter, the index entry from that <X> tag would reference page 2-132.

The <X> tag has the following syntax:

```
<X> (index-entry[\attribute])
```

Use the *index-entry* argument to supply the text for the main entry and subentries in the index. The text used in this argument is placed in the index entry exactly as entered, including any capitalization. Use the <XSUBENTRY> tag within this argument to separate the main entry from the first subentry, and the second subentry from the first, as follows:

```
<X> (Main entry <XSUBENTRY> subentry-1 <XSUBENTRY> subentry-2)
```

The <XSUBENTRY> tag can be abbreviated as <XS> . You can use no more than three subentries in an index.

Use the *attribute* arguments to control the sorting and formatting of the index entry. You can specify up to five attribute arguments to the <X> tag. Section 7.2.2 describes the indexing tag attributes.

Use the following <X> tag coding rules to ensure that the page references in your main index entries are correct:

- Place <X> tags after tags in your SDML file that are likely to begin a new page, such as <CHAPTER> or <HEAD1> .
- Index entries that are to be merged in the index should be coded identically in the SDML file, including emphasis, capitalization, spacing and spelling.
- When inserting <X> tags into tables, place the <X> tags within the arguments to the <TABLE_ROW> tag, so that the page references will be correct if the text formatter breaks the table across pages.

- Do not place <X> tags within examples. Doing so may interfere with the formatting of the examples.

The following example shows the correct positioning of the <X> tag that provides an indexing entry for a chapter:

```
<CHAPTER>(Creating Routines\creating_routine_sec)
<X>(Routines<XSUBENTRY>creation of)
<P>
In order to create a routine, you must first
```

If the previous text appeared on page 11-2, this <X> tag would create the following index entry:

Routines

creation of, 11-2

7.2.2 Using Indexing Tag Attributes

The *attribute* arguments to the <X> and <Y> tags control the sorting and formatting of individual index entries. These attributes are described alphabetically in the following sections. All of these attributes can be used with the <X> tag; however, only the MASTER and <XSORT> attributes can be used with the <Y> tag.

7.2.2.1 Using the BEGIN and END Attributes

Use the BEGIN and END attributes to create an index entry for consecutive pages. These attributes create an index entry with a page range such as "System Concepts, 12-3 to 12-7." When you use a BEGIN and END pair, the index entries and indexing attributes you specify must be identical for the index entries to be sorted correctly.

The following example shows how a page range can be coded using the BEGIN and END attributes. In this example, the tag with the BEGIN attribute is processed on page 12-3 and the tag with the END attribute is processed on page 12-7.

```
<HEAD1>(Concepts of the System\sys_con)
<X>(System Concepts\BEGIN)
<P>
 .
 .
 .
<X>(System Concepts\END)
<HEAD1>(Using the System\sys_using)
```

These index tags would create a page range as follows:

System Concepts, 12-3 to 12-7

<X> tags that occur between a BEGIN and END pair that have entry text identical to that of the BEGIN and END pair are ignored, and VAX DOCUMENT issues an informational message stating that this index entry was ignored when it creates the index.

For example, if an index entry for "System Concepts" was processed on output page 12-4, between the index entries specified with the BEGIN and END attributes in the previous example, the code would appear as in the following example:

```
<X>(System Concepts\BEGIN)
.
.
.
<X>(System Concepts)
<X>(System Concepts<XS>entering files)
.
.
.
<X>(System Concepts\END)
```

These index tags would create the following index entry:

System Concepts, 12-3 to 12-7
  entering files, 12-4

The <X> (System Concepts) tag on page 12-4 would be ignored, and VAX DOCUMENT would issue an informational message about that tag when it creates the index. Note that the <X> tag whose entry text included the subentry "entering files" was not ignored during the creation of the index.

7.2.2.2 Using the BOLD Attribute

Use the BOLD attribute to cause page numbers to appear in bold type in your index. If both the BOLD and ITALIC attributes are specified, the page number will be output in bold italic type.

Page number references that are output in a bold type face using the BOLD attribute are sorted as being distinct from page number references that are output in a bold type face using some other means (such as the global <EMPHASIS> (\BOLD) tag).

The following is an example of an <X> tag that uses the BOLD attribute:

```
<X>(System Concepts\BOLD)
```

If this index tag occurred on page 12-3, it would create an entry in the index as follows:

System Concepts, **12-3**

7.2.2.3 Using the ITALIC Attribute

Use the ITALIC attribute to cause page numbers to appear in italic type in your index. If both BOLD and ITALIC are specified, the page number will be output in bold italic type.

Page number references that are output in an italic type face using the ITALIC attribute are sorted as being distinct from page number references that are output in an italic type face using some other means (such as the global <EMPHASIS> tag).

The following is an example of an <X> tag that uses the ITALIC attribute:

```
<X>(System Formats\ITALIC)
```

If this index tag occurred on page 6-11, it would create an entry in the index as follows:

System Concepts, *6-11*

7.2.2.4 Using the MASTER Attribute

Use the MASTER attribute to specify an index entry that you want to appear in a master index. The following is an example of an <X> tag that uses the MASTER attribute:

```
<X>(System formats\MASTER)
```

By default, entries labeled with the MASTER attribute are included only in the master index and entries not labeled with the MASTER attribute are included only in the index of a single document.

You can override these defaults by using the following DOCUMENT command line qualifiers:

/INDEX=OVERRIDE_MASTER	Creates a single document index containing both master and nonmaster index entries.
/MASTER_INDEX=OVERRIDE_MASTER	Creates a master index containing both master and nonmaster index entries.

See Section 7.2.5 for more information on the OVERRIDE_MASTER keyword. See Section 7.3 for information on creating a master index.

7.2.2.5 Using the <XAPPEND> Attribute

Use the <XAPPEND> attribute to append a specified string to the end of a main entry page reference. You append the string by passing that string as an argument to the <XAPPEND> attribute as <XAPPEND> (string). Use the <XAPPEND> attribute to add information to a main index entry, for example, to mark an index page as referring to a table (9-8tab) or to insert common indexing terms such as "page 3ff" or "5-3 to 5-6 passim."

The following example shows an <X> tag that appends the string "ff" to the page reference using the <XAPPEND> attribute:

```
<X>(File structure\<XAPPEND>(ff))
```

If this tag were processed on page 3, this tag would create the following main index entry:

File structure, 3ff

7.2.2.6 Using the <XSORT> Attribute

Use the <XSORT> attribute to control the default sorting of individual index entries. For example, if you wanted the entry "$$" to appear in your index sorted as the words "double dollar," you could place the <XSORT> tag in your SDML file as follows:

```
<X>($$\<XSORT>(double dollar))
```

This attribute is especially useful for overriding the default sorting of leading nonalphabetic characters in individual index entries.

7.2.3 Creating Cross-Reference Index Entries

The <Y> tag creates an entry in an index file that has no output page associated with it. This tag is typically used to create cross-references to other index entries ("See" or "See also" entries). The <Y> tag has the following syntax:

<Y> (index-entry[\attribute])

The <Y> tag accepts the same *index-entry* argument as the <X> tag. Index subentries are specified within the *index-entry* using the <XSUBENTRY> (abbreviated as <XS>) tag just as for the <X> tag. The <Y> tag accepts only the MASTER and <XSORT> indexing tag attributes because all other attributes are for manipulating page numbers.

MASTER indicates that the index entry should only occur in the master index and <XSORT> lets you override the sorting algorithm for an individual entry. See Section 7.2.1 for more information on these attibutes. Section 7.2.2 describes the indexing tag attributes.

The following example shows the <Y> tag used with the <X> tag. The <X> tag creates an index entry with a page number reference and the <Y> tag creates a "See also" index entry:

```
<X>(File structure designators)
<Y>(File structure designators<XSUBENTRY>See also Header blocks)
```

These indexing tags create the following index entry (assuming these tags appeared on page 1-3):

```
File stucture designators, 1-3
    See also Header blocks
```

7.2.4 Processing an Index

When you use the DOCUMENT /INDEX qualifier, VAX DOCUMENT processes the index entries you created in your SDML file, and places them in a separate index file. VAX DOCUMENT names this file by taking the file name of the *input-file-spec* parameter you specified on the DOCUMENT command line and appending "_INDEX" to it.

If you omit text formatting by using the /NOTEXT_FORMATTER qualifier, index processing will also be omitted, because the index is processed during text formatting. If you use the /INDEX and /NOTEXT_FORMATTER qualifiers together, VAX DOCUMENT issues an error message.

You need do nothing more than specify /INDEX on the command line if you want to maintain your index in a separate file; however, if you want this file automatically incorporated into your document, place the <INDEX_FILE> tag in your SDML file.

When the <INDEX_FILE> tag is processed by VAX DOCUMENT, the latest index file is incorporated into the final output file where the <INDEX_FILE> tag occurs in the SDML file. If you use the /NODEVICE_CONVERTER qualifier, the index will not be incorporated into your file.

Use the /INDEX qualifier whenever you process a file that contains the <INDEX_FILE> tag to make sure that the latest version of the index is incorporated into your file. This action ensures that the index file correctly reflects the organization and pagination of your SDML file. If you do not specify /INDEX, an out of date index may be included into your document.

When you create an index using the /INDEX qualifier, whatever parameters and other qualifiers you specify affect the processing of both the index file and your SDML file. For example, if you specify LN03 as a destination and also use the /PRINT qualifier, both files will be printed on the LN03 laser printer (*input-filename*.LN03 and *input-filename*_INDEX.LN03).

The following example shows how an index may be created and printed using the /INDEX and /PRINT qualifiers. Both the SDML file (MYREPORT) and the index file created from that SDML file (MYREPORT_INDEX) are formatted for the LN03 laser printer and printed on it as specified on the command line. VAX DOCUMENT concatenates the printing of the two files into a single print job so VMS issues only one print message.

```
$ DOCUMENT myreport.sdml REPORT LN03 /INDEX /PRINT=(NOTIFY)

%DOC-I-IDENT, VAX DOCUMENT 1.1

[ T a g   T r a n s l a t i o n ]...
%TAG-I-DEFSLOADD, End of Loading of Tag Definitions
%TAG-I-ENDPASS_1, End of first pass over the input
[ T e x t   F o r m a t t i n g ]...
%TEX-I-PAGESOUT, 17 pages written.
-TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYREPORT.DVI_LN03'
[ I n d e x   G e n e r a t i o n ]...

%INX-I-ENDPASS_1, End of first pass over input file:
   'DUA1:[DOCFILES]MYREPORT_INDEX.INX'
%INX-I-ENDPASS_2, End of second pass over input file.

%INX-I-CREATED, 'DUA1:[DOCFILES]MYREPORT_INDEX.TEX;1' created
[ T e x t   F o r m a t t i n g   I n d e x ]...
%TEX-I-PAGESOUT, 1 page written.
-TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYREPORT_INDEX.DVI_LN03'
[ D e v i c e   C o n v e r s i o n ]...
%DVC-I-PAGESOUT, 18 pages written to file:
   DUA1:[DOCFILES]MYREPORT.LN03
[ I n d e x   D e v i c e   C o n v e r s i o n ]...
%DVC-I-PAGESOUT, 1 page written to file:
   DUA1:[DOCFILES]MYREPORT_INDEX.LN03
[ P r i n t i n g   F i l e ]...
[ P r i n t i n g   I n d e x ]...

Job MYREPORT (queue SYS$LN03, entry 835) started on SYS$LN03

$
```

7.2.5 Using Indexing Options

The following subsections describe the optional keywords that can be used with the /INDEX and /MASTER_INDEX qualifiers; the keywords have the same meanings for both qualifiers except where noted.

These keywords allow you to control whether guide headings are used, how master index entries should be processed, and how index entries should be sorted. The indexing option keywords are as follows:

- [NO]GUIDE_HEADINGS
- [NO]OVERRIDE_MASTER
- SORT

These keywords are described alphabetically in the following subsections, and are specified as follows:

/INDEX[=(index-keyword [,index-keyword...])]
/MASTER_INDEX[=(index-keyword [,index-keyword...])]

7.2.5.1 Using the GUIDE_HEADINGS and NOGUIDE_HEADINGS Keywords

Use the GUIDE_HEADINGS and NOGUIDE_HEADINGS keywords to specify whether alphabetic headings are inserted into the index whenever an entry beginning with a new letter occurs. The GUIDE_HEADINGS keyword is the default.

For example, when the first index entry beginning with the letter "B" is encountered, the guide heading "**-B-**" is placed before it.

NOGUIDE_HEADINGS suppresses guide headings in the index output file.

7.2.5.2 Using the OVERRIDE_MASTER and NOOVERRIDE_MASTER Keywords

The OVERRIDE_MASTER and NOOVERRIDE_MASTER keywords specify how index entries with and without the MASTER attribute are included in a single document index and a master index.

- Use /INDEX=NOOVERRIDE_MASTER to create a single document index that contains only the index entries that are not marked with the MASTER keyword; this is the default.
- Use /INDEX=OVERRIDE_MASTER to create a single document index that contains both the index entries that are not marked with the MASTER keyword and the entries that are marked with the MASTER keyword.
- Use /MASTER_INDEX=NOOVERRIDE_MASTER to create a master index that contains only the index entries that are marked with the MASTER keyword; this is the default.
- Use /MASTER_INDEX=OVERRIDE_MASTER to create a master index that contains both the index entries that are not marked with the MASTER keyword and those entries that are marked with the MASTER keyword.

7.2.5.3 Using the SORT Keyword

Use the SORT keyword to specify the sorting algorithm used to order entries in an index. The SORT keyword has the following syntax:

SORT[=sort-keyword[,sort-keyword[=value]]]

The following are valid sort-keywords:

- SORT=LETTER sorts the entries letter-by-letter and ignores spaces and hyphens. SORT=LETTER is the default.

  For example the entry "A directory" would be sorted after the entry "Add," because the space is ignored in the sorting and so "A di" is placed after "Add" by the sort.
- SORT=WORD sorts the entries letter-by-letter and treats spaces and hyphens as significant.

For example the entry "A directory" would be sorted with other entries that began with "A d" rather than with the entry "Add" because the space is treated as a significant part of the entry during the sorting; this results in "A d" being placed before "Add" by the sort (because by default nonalphabetic characters are sorted before alphabetic characters).

- SORT=NONALPHA positions entries with initial nonalphanumeric characters in the index based on the following keyword, supplied with the NONALPHA keyword:
  - The AFTER keyword causes entries with initial nonalphanumeric characters to be placed at the end of the index.
  - The BEFORE keyword causes entries with initial nonalphanumeric characters to be placed at the beginning of the index.
  - The IGNORE keyword causes entries with initial nonalphanumeric characters to be sorted by the first alphanumeric characters in the entry. The default is NONALPHA=IGNORE.

7.3 Creating a Master Index

VAX DOCUMENT lets you combine the indexes from several documents to create a master index for a documentation set. This master index has the same format as that of a single-document index, with the following additions:

- Master index entries are cited both by the title of the document from which they originated and by page number. Document titles are italicized in these index entries.
- Index entries that were specified using the MASTER keyword attribute in individual documents are automatically entered into the master index.

You can modify your master index by specifying indexing option keywords. The /MASTER_INDEX qualifier accepts the same indexing option keywords as the /INDEX qualifier. See Section 7.2.5 for more information on the indexing option keywords.

You create a master index by performing the following steps:

1. Create individual intermediate index (INX) files using the /INDEX and /KEEP=(INX) qualifiers to the DOCUMENT command.
2. Create a master index data file, which lists each of the intermediate index files with the title of the book from which the index file was generated.
3. Run VAX DOCUMENT using the /MASTER_INDEX qualifier and specifying the master data file as the *input-file-spec* parameter.

   This will produce a final printable master index file with the same file name as the master data file and with a file type associated with the destination keyword specified on the command line (for example, LN03).

7.3.1 Creating Intermediate Index Files

You create intermediate index files whenever you use the /INDEX qualifier to create an index. However, VAX DOCUMENT deletes these files unless you also specify the /KEEP specifier with the INX keyword argument. You can specify these qualifiers, as in the following example:

```
$ DOCUMENT mychapter.sdml REPORT LN03 /INDEX /KEEP=(INX)
```

7.3.2 Creating the Master Index Data File

You create the master index data file just as you would create any ASCII file (by using an editor, using the VMS CREATE command, and so on). The master index data file lists all the index files that are to be collated into the master index. Give the master index data file a unique file name. When you use the master index data file as the source file for your master index, the DOCUMENT command will expect a default file type of INX_LIST.

Each master index entry is associated with the book that it came from. You can use the /BOOK_IDENTIFIER qualifier inside of the data file to specify the title of that book. If you do not specify the /BOOK_IDENTIFIER qualifier, the title of the book will be the file name of the INX file. The /BOOK_IDENTIFIER qualifier has the following syntax:

/BOOK_IDENTIFIER="book-title"

If you specify the /BOOK_IDENTIFIER without the *book-title* value, VAX DOCUMENT issues an error message. If you want to generate a master index without book titles, you can specify null strings to the /BOOK_IDENTIFIER qualifier for all your intermediate index files as follows:

```
2041.INX/BOOK_IDENTIFIER=" "
2323.INX/BOOK_IDENTIFIER=" "
```

Use the following rules in entering intermediate index file specifications (*filename*.INX files) into the master index data file:

- Each index file specification must be entered on a single line in the data file with no other characters preceding it.
- Each index file specification must be an intermediate indexing file (*filename*.INX) and must be specified completely including the INX file type. You can also use a process logical name for the intermediate indexing file.
- Comments can be made in the data file using the exclamation point (!) character. This character excludes from processing only those characters that follow it on the same line.

The following is a sample of a master index data file. Because the books listed in this file all have numeric names, the /BOOK_IDENTIFIER qualifier was used to specify more meaningful book titles.

```
!Master index data file
!Created Dec 1, 1986
!Updated Jan 4, 1987
!
    2041.INX/BOOK_IDENTIFIER="System Generation"
    2323.INX/BOOK_IDENTIFIER="Writing I/O Driver"
    2050.INX/BOOK_IDENTIFIER="MCR Operations"
    2053.INX/BOOK_IDENTIFIER="Program Development"
    2054.INX/BOOK_IDENTIFIER="Executive"
    2055.INX/BOOK_IDENTIFIER="Task Builder"
    2056.INX/BOOK_IDENTIFIER="System Library"
    2057.INX/BOOK_IDENTIFIER="Utilities"
    2059.INX/BOOK_IDENTIFIER="I/O Drivers"
    2176.INX/BOOK_IDENTIFIER="Release Notes"
```

7.3.3 Creating the Master Index File

You create the printable master index file by running the DOCUMENT command with the /MASTER_INDEX qualifier and specifying the master index data file as the *input-file-spec* parameter. The master data file has a default file type of INX_LIST.

In the following example, the master index data file MYMASTER.INX_LIST is processed using the /MASTER_INDEX qualifier to create the file MYMASTERINDEX.LN03:

```
$ DOCUMENT MYMASTER.INX_LIST REPORT LN03 /MASTER_INDEX
%DOC-I-IDENT, VAX DOCUMENT V1.1
[ M a s t e r   I n d e x   G e n e r a t i o n ]...
%INX-I-ENDPASS_1, End of first pass over input file:
   'DUA1:[DOCFILES]2041.INX'
%INX-I-ENDPASS_2, End of second pass over input file.
%INX-I-ENDPASS_1, End of first pass over input file:
   'DUA1:[DOCFILES]2323.INX'
%INX-I-ENDPASS_2, End of second pass over input file.
%INX-I-ENDPASS_1, End of first pass over input file:
   'DUA1:[DOCFILES]2050.INX'
%INX-I-ENDPASS_2, End of second pass over input file.
%INX-S-CREATED, 'DUA1:[DOCFILES]MYMASTER.TEX;1' created
[ T e x t   F o r m a t t i n g ]...
%TEX-I-PAGESOUT, 8 pages written.

-TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYMASTER.DVI_LN03'
[ D e v i c e   C o n v e r s i o n ]...
%DVC-I-PAGESOUT, 18 pages written to file:
  DUA1:[DOCFILES]MYMASTER.LN03
[ P r i n t i n g   F i l e ]...

Job MYMASTER (queue SYS$LN03, entry 835) started on LN03

$
```

8 Special Features

This chapter describes various special features that you can request in the formatted output. The features include the following:

- Providing emphasis, such as bolding or italicizing
- Using footnotes
- Using callouts
- Drawing large braces and brackets
- Controlling case
- Formatting mathematical formulas
- Providing quotation marks
- Placing parentheses around a single character
- Drawing horizontal and vertical ellipses

8.1 Providing Emphasis

There are numerous methods available to lend emphasis to words or phrases in your printed output. In most methods, you pass the text to be emphasized as an argument to a special tag.

This section discusses the tags as global tags, because they are always available for use. The formatted effect of each tag is also described here, but you should be aware that the effect can differ depending on the doctype.

The <EMPHASIS> tag provides italic or bolding emphasis. At first glance, those capabilities would seem to be adequate for most purposes, but in fact there are many additional tags that provide emphasis. The variety of tags relates to the numerous possible reasons for emphasis.

The reasons vary: the word might be the name of something, it might be a reserved word in a programming language, or it might be being used for the first time in your book. In a generic markup language, you should be tagging the word according to the reason for its emphasis, not according to the desired effect in the formatted output. In other words, you should be thinking, "This is the first use of this phrase," and not thinking, "This phrase needs to be italicized."

Accordingly, the tags for emphasis attempt to name the reason for the emphasis rather than to name the formatting effect. If you use the tags properly in your SDML files, the book designer is free to change the effect for a single tag. Thus, the tag that lends emphasis to a first-time use, <NEWTERM> , might be changed from simple italic to an italic font that is larger or bolder. However, another tag that also gives an italic effect might be left unchanged. If you fail to distinguish these two uses when you code the SDML file (by coding both with the <EMPHASIS> tag), you will not be able

to make a selective change in the type of emphasis for the <NEWTERM> tag without affecting your output.

A description of each of the emphasis tags follows:

- <GREF> and <NEWTERM>

  These two tags are similar, in that they are used to refer to a word or phrase in a special first-time use, where the word is about to be defined. Use the <GREF> tag if the word is also in the glossary.

- <KEYWORD>

  Use this tag in a discussion of a word or phrase from a computer program or a programming language (sometimes called a "reserved word").

- <VARIABLE>

  When you write about the effect of executing some computer program command or programming language statement, you may want to refer to a variable by name and then refer to its value. Tagging the variable name with this tag ensures that it has the proper emphasis in the output.

- <EMPHASIS> and <UNDERLINE>

  These tags are appropriate for situations where the other emphasis tags are unsuitable. Because they specify the formatting effect in the output rather than the reason for the effect, use them sparingly.

- <NOTE>

  This tag has a pronounced formatting effect, because it not only bolds the text, but it breaks the text out of its surrounding text and supplies an attention-getting heading. An example of an appropriate use follows:

```
. . .  the unary minus in this expression.
<NOTE>Observe how the precedence of the unary minus operator
requires the use of parentheses in this expression.
<ENDNOTE>
```

  This example might produce the following:

  . . . the unary minus in this expression.

Note: Observe how the precedence of the unary minus operator requires the use of parentheses in this expression.

Table 8–1 shows each of the tags (except <NOTE>) used in text. Your output might differ, according to the design chosen for your system.

Table 8–1 Examples of Emphasis Tags

Coding	Output[1]
The <GREF> (abc def) letters	The *abc def* letters
The <NEWTERM> (abc def) letters	The *abc def* letters
The <KEYWORD> (abc def) letters	The **abc def** letters
The <VARIABLE> (abc def) letters	The *abc def* letters
The <EMPHASIS> (abc def) letters	The *abc def* letters
The <EMPHASIS> (abc def\BOLD) letters	The **abc def** letters
The <EMPHASIS> (abc def\SMALLCAPS) letters	The ABC DEF letters
The <EMPHASIS> (abc def\SMALL_BOLDCAPS) letters	The **ABC DEF** letters
The <UNDERLINE> (abc def) letters	The abc def letters

[1]The output of the samples included in this table is dependent on the doctype and printing device used.

As you read about each tag, remember that the punctuation that follows a bolded or italic font should be printed in the same font. This means that you should always include the punctuation as part of the argument.

8.2 Using Footnotes

Three tags are available for creating a footnote, the <FOOTNOTE> [1]tag, the <FOOTNOTE_TEXT> tag, and the <FOOTREF> tag. Their syntax is as follows:

```
<FOOTNOTE>(char\footnote-text)
<FOOTNOTE_TEXT>(char\text)
<FOOTREF>(char-1[\char-2...\char-9])
```

A footnote consists of two parts, the character that appears as a small superscript in the text, and the correspondingly noted sentence or paragraph at the foot of the same page. The <FOOTNOTE> tag provides both the superscript and the actual text.

You supply the superscript character as the first argument of the <FOOTNOTE> tag and the footnote text as its second argument. Place the <FOOTNOTE> tag precisely where you want the number to appear. This is generally at the end of a phrase or sentence and usually after the closing punctuation. Do not leave space between the punctuation and the opening angle bracket of the <FOOTNOTE> tag.

VAX DOCUMENT places the character as a small superscript in the text, and floats the text to the foot of the page.

You can use the <FOOTREF> tag to place the same number at another place in the text, although this practice is discouraged.

Use the <FOOTNOTE> and <FOOTREF> tags to create footnotes in tables. Table footnotes appear at the foot of the table, rather than at the foot of the page. When you need a footnote in a table, place the <FOOTNOTE> tag at the beginning of the table immediately after the <TABLE_SETUP> tag. Then, at the desired points within the table, supply the <FOOTREF> tag. With this method, you can place the same superscript at more than one point in the table.

For multipage tables, the footnote text repeats on each page that carries a reference to that note, provided that you call the footnotes out in the table heading. If the reference is in the heading, the footnotes and references appear on all pages of the table.

The output device must have a character font that displays these superscript characters.

The <FOOTNOTE_TEXT> tag is necessary when a footnote must be provided on a title or copyright page or in the context of header-level text.

8.3 Using Callouts

Three tags are available for annotating examples by what is termed a callout. The callout is a number that draws special attention, possibly by being printed as a white digit on a black background, for example ❷, or through some other graphic display. Writers frequently display numbers this way to mark points in an example that are then explained by entries in a numbered list that follows the example. The numbers of the list entries are displayed in the same format (white digit on a black background).

Callouts are used within a restricted area of the SDML file that you indicate by <CALLOUTS> and <ENDCALLOUTS> tags. Within the example or text that is to be annotated, you locate the desired numbers by using the <CO> tag with the correct number as its argument. Following the area of the callouts, you supply the list of explanations or notes by introducing the list with the <LIST> (CALLOUT) tag. The list entries are introduced by the <LE> tag and the list is ended with the <ENDLIST> tag. However, when the list is formatted, the number of each entry in a noted-list is displayed as a white number on a black background.

When you begin an example that will include callouts, you can specify \PREFIX as an argument of the <CALLOUTS> tag. This causes the example to be indented, leaving a gutter on the left. Then, if you specify the <CO> tag at the left end of an example line, the callout number will be placed in the left gutter. In this way, VAX DOCUMENT does not disturb the normal indentation that might be important in your example. Here are two identical example lines, coded as follows:

```
<CO>(2)PROGRAM Calculator (INPUT, OUTPUT);
TYPE
  Yes_No = (Yes, No);
```

Without the \PREFIX argument, the output format is as follows:

```
❷PROGRAM Calculator (INPUT, OUTPUT);
TYPE
  Yes_No = (Yes, No);
```

With the \PREFIX argument, the output format is as follows:

```
❷ PROGRAM Calculator (INPUT, OUTPUT);
  TYPE
    Yes_No = (Yes, No);
```

Callouts are not limited to use within monospaced examples. For example, if you want to number certain paragraphs of text, or if you want to place callouts into a template to distinguish one group of entries from another, you can surround the area with the <CALLOUTS> ... <ENDCALLOUTS> tags and then place the <CO> tags wherever numbers are desired. For text paragraphs, you must be sure to place the <CO> tag after the <P> tag

that starts a paragraph, and in other cases you may have to place the <CO> tag within an argument list so that the callout number is associated with the proper text elements. If you use the \PREFIX argument on the <CALLOUTS> tag when you put callouts on paragraphs, the paragraphs are not indented, but the callout number is pushed to the left of the normal text margin, as follows:

❶ This is a paragraph with a callout. The \PREFIX argument was used on the <CALLOUTS> tag.

If you need to refer to a noted-list entry, you should use the <CALLOUT_REF> (callout-number) tag. This tag need not occur within the <CALLOUTS> . . . <ENDCALLOUTS> tags. The word "callout" is editorial jargon and is probably not in your reader's vocabulary, so do not write "see callout ❸." Instead, refer to the numbered item as a note. For example, "see note ❸."

Each device treats callouts in a distinct way typographically. This might vary from device to device according to the available fonts.

8.4 Drawing Large Braces and Brackets

When you are documenting the syntax of a programming language, you often use the convention that shows choices (or options) as items that are vertically stacked inside large brackets or braces. You tell the reader that the brackets mean an optional choice and the braces mean a required choice.

These brackets or braces are produced in the output by using the <LIST> (STACKED) tag. The <LIST> (STACKED) tag works like other list tags in that the list begins with the <LIST> tag, each element is introduced by the <LE> tag, and the list is ended by the <ENDLIST> tag. The size of the bracket or brace is adjusted automatically to enclose the list elements and the enclosed list is aligned with other elements on the same line. Here is an SDML file excerpt:

```
<P>
SET
<LIST>(STACKED\BRACES)
<LE>LOZENGE
<LE>UNDERSCORE
<LE>GRAPHIC
<ENDLIST>
CURSOR
<LIST>(STACKED\BRACKETS)
<LE>ON
<LE>OFF
<ENDLIST>
(Default is ON)
```

The result is as follows:

SET $\left\{\begin{array}{l}\text{LOZENGE}\\ \text{UNDERSCORE}\\ \text{GRAPHIC}\end{array}\right\}$ CURSOR $\left[\begin{array}{l}\text{ON}\\ \text{OFF}\end{array}\right]$ (Default is ON)

The output device must have a character font that displays these braces and brackets.

8.5 Controlling Case

Usually, you control the case of the letters in the output by the case of the letters in the SDML file. If you put a lowercase letter in the SDML file, it remains lowercase in the output. In rare cases, an argument to a tag is forced into upper- or lowercase by the action of the tag. If you want to avoid this case conversion, you can force letters into the output (in the stated case) by using either the <UPPERCASE> or <LOWERCASE> tags.

8.6 Providing Quotation Marks

Many keyboards provide a key for a quotation mark, sometimes called a double-quote, that produces an effect such as shown in the following example:

```
"Hello, sailor!"
```

The quotation marks used by typesetters, on the other hand, produce an effect such as the following:

“Hello, sailor!”

In software documentation, you might want the programmer's double-quote character ("). Occasionally, you will also want to surround something with the typesetter's quotation marks (“ ”).

When you want the typesetter's quotation marks around a string, pass the quoted string as an argument to the <QUOTE> tag. For example, to obtain “Hello, sailor!” you must specify the following:

```
<QUOTE>(Hello, sailor!)
```

Include ending punctuation as part of the argument to the <QUOTE> tag if you want it to appear inside the quotation marks.

VAX DOCUMENT outputs one of the typesetter's quotation marks (the ”) when it finds the ASCII double-quote character (") in a proportionally spaced part of your SDML file. However, in an example, where monospaced text is the default, the ASCII double-quote character is output as (").

If you need to show the ASCII double-quote character in text as it normally appears in a listing or on the screen (as "), code it with the <DOUBLE_QUOTE> tag. Suppose you want to show the following:

```
. . .  in the expression NAME := "Smith", . . .
```

You must code this text as follows:

```
...in the expression NAME := <DOUBLE_QUOTE>Smith<DOUBLE_QUOTE>,...
```

Similarly, the text output for a single-quote character (ASCII 39) is the apostrophe ('), so if you need to show the ASCII single-quote character in text as it normally appears in a listing or on the screen (as '), code it with the <SINGLE_QUOTE> tag. Suppose you want to show the following:

```
. . .  in the expression NAME := 'Smith', . . .
```

You must code this text as follows:

```
...in the expression NAME := <SINGLE_QUOTE>Smith<SINGLE_QUOTE>,...
```

This might appear complicated, but you often achieve what you want with minimum effort. The rules are as follows:

1 In examples, where you are illustrating what appears on the screen, use single- or double-quote characters. Just press the keyboard's single- or double-quote key. Because it is within an example, the key results in the screen's single- or double-quote character (' or ") rather than the apostrophe or the typesetter's quotation mark.

2 In text, when you want the apostrophe, press the keyboard's single-quote key. Because it is within text, it prints as an apostrophe.

3 In text, when you need to cite text from another source or cite an excerpt of a speech, use the <QUOTE> tag, to obtain the typesetter's quotation marks.

4 You will rarely want to show the screen or listing version of the single-quote key (') or the double-quote key (") in text. However, when necessary, use the <SINGLE_QUOTE> and <DOUBLE_QUOTE> tags.

The output device must have a character font that displays these typesetter's characters.

8.7 Placing Parentheses Around a Single Character

When you enclose something in parentheses in your text, the opening and closing parentheses snug up against the enclosed text. However, if you are enclosing a single character, especially a punctuation character, the close placement of the parentheses overwhelms the character, for example (,). If you anticipate this possible problem and place a space on either side of the character, (,), you run the risk that a line break will occur at one of the spaces, leaving some of the parenthesized character at the end of one line and the rest at the start of the next line.

Instead, you can pass the character as an argument to the <PARENDCHAR> tag, which will supply the parentheses and some extra space around the character, for example, (,). The argument to <PARENDCHAR> need not be limited to one character, but if the characters being parenthesized are letters or digits the <PARENDCHAR> tag may be unnecessary.

The output device must have a character font that displays these proportionally spaced characters.

8.8 Drawing Horizontal and Vertical Ellipses

Writers sometimes use ellipses to emphasize the omission of text. It is not possible to achieve proper spacing of the ellipses by simply supplying three periods, because periods snug up against the preceding character. If you insert space characters between them, you run the risk that a line break will occur between them. Accordingly, use the <HELLIPSIS> tag to supply a horizontal ellipsis (. . .).

To create a vertical ellipsis, use the <ELLIPSIS> tag.

8.9 Showing Special Characters

VAX DOCUMENT provides facilities for including special characters in the text. Among the special characters supported are the following:

- Mathematical special symbols, such as operators and the uppercase Greek letters
- Multinational Character Set (MCS), which includes such symbols as the British pound, the Japanese yen, umlauts, and so forth
- Opening double brackets and closing double brackets

The <MATH_CHAR> tag allows you to specify a wide range of mathematical symbols. The <MCS> tag offers the full range of symbols in the DIGITAL Multinational Character Set. The <SPECIAL_CHAR> tag supports a series of keywords that identifies such special symbols as the trademark symbol, the dagger, and the opening double bracket and closing double bracket.

8.10 Using Formatting Tags

The following tags provide special formatting:

- <PAGE>
- <LINE>
- <KEEP>
- <FINAL_CLEANUP>

Be aware when using these tags that your output can vary between devices. If you were to use these tags in a file processed for an LN01 printer, your output could look much different than you planned if you were to reprocess your file for a POSTSCRIPT device. Adding or deleting text can also affect the output when these tags are used.

8.10.1 Specifying Page Breaks

Use the <PAGE> tag whenever you need to force the text to begin on a new page of output. You can optionally direct whether the text continuation should begin on the next even- or odd-numbered page.

8.10.2 Specifying Line Breaks

Use the <LINE> tag whenever you must force text to begin on a new line of output. It is possible to include arguments that determine the amount of space before the next line or whether the text should be indented.

In addition, the <CENTER_LINE> tag identifies a line of text that is to be centered within the current text margins. Its arguments also permit you to specify the amount of space introduced prior to outputting the line.

The <RIGHT_LINE> tag right-adjusts a line of text within the current text margins. Optionally, you can specify that a large or small amount of space is desired.

8.10.3 Keeping Text on a Single Line

The <KEEP> tag allows you to specify text that must be forced to occur on a particular line of output and, therefore, must not be either hyphenated or broken at an existing hyphen. For example, this is important whenever it is undesirable to break a hyphenated word, such as a product name like PDP–11 or when a widow or orphan might be created by automatic hyphenation.

Note that when you use the <KEEP> tag in combination with the <HYPHENATE> tag, the hyphenation does not occur; that is, the <KEEP> tag overrides the normal effects of the <HYPHENATE> .

8.10.4 Controlling Page and Line Breaks for Final Production

The <FINAL_CLEANUP> tag is useful for performing specialized adjustments of line or page breaks during final production, when the use of a particular output device or some other consideration indicates that the normal page or line break should be overridden.

9 Doctype-Independent Tag Descriptions

This chapter includes descriptions of all doctype-independent (globally used) tags. The tags are listed alphabetically.

<ABSTRACT>

Begins a summary description of a document on the title page or part page of a document.

FORMAT

<ABSTRACT> *[(title)]*

ARGUMENTS

title
Specifies a title for the abstract.

related tags

- <FRONT_MATTER>
- <TITLE_PAGE>
- <PART_PAGE>

restrictions

Can be used only in the context of a <TITLE_PAGE> or <PART_PAGE> tag.

required terminator

<ENDABSTRACT>

DESCRIPTION

The <ABSTRACT> tag begins a summary description of a document on the title page or part page of a document.

EXAMPLES

See the examples in the discussion of the <PART_PAGE> tag.

<ACCENT>

Supplies a freestanding accent mark.

FORMAT

<ACCENT> *(accent-mark)*

ARGUMENTS

accent-mark

Specifies one of the following keywords to indicated the accent mark:

- UMLAUT
- ACUTE
- GRAVE

related tags

- <MCS>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <ACCENT> tag supplies a freestanding accent mark in the printed output.

EXAMPLE

```
<P>Accents used in the DEC Multinational Character Set include the
following:
<LIST>(UNNUMBERED)
<LE>Grave accents <PARENDCHAR>(<ACCENT>(GRAVE))
<LE>Acute accents <PARENDCHAR>(<ACCENT>(ACUTE))
<LE>Umlauts <PARENDCHAR>(<ACCENT>(UMLAUT))
<ELS>
```

This example may produce the following output:

Accents used in the DEC Multinational Character Set include the following:

- Grave accents (`)
- Acute accents (´)
- Umlauts (¨)

<ALIGN_AFTER>

Allows you to position formatted text in a list.

FORMAT

<ALIGN_AFTER> *(text)*

ARGUMENTS

text
Specifies the string under which you want to align a stack of text.

related tags

- <LINE> [(INDENT)]
- <LIST> (STACKED)
- <ALIGN_NUMBER>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <ALIGN_AFTER> tag enables you to position text in a list after the text specified in the argument.

EXAMPLE

```
<LIST>(STACKED\BRACES)
<LE><ALIGN_AFTER>(UN)NUMBERED
<LE>UNNUMBERED
<ELS>
```

This example shows how to create a stacked list where the text is aligned after the the string "UN." This example may produce the following output:

$$\left\{\begin{array}{r} \text{NUMBERED} \\ \text{UNNUMBERED} \end{array}\right\}$$

<ALIGN_CHAR>

Identifies a nonprinting character to be used to align numeric information within a column of a list or table.

FORMAT

<ALIGN_CHAR> *(character [\ DELTA])*

ARGUMENTS

character

Specifies the character to be used for alignment.

The character must be one of the following special characters:

{ } ^ # $

% * ~ @

[ ] < >

DELTA

Specifies the character indicated for alignment is replaced on output with a delta character to indicate spacing.

related tags

- <ALIGN_AFTER>
- <TABLE_ROW>
- <ALIGN_NUMBER>

restrictions

Invalid in math.

Invalid in monospaced examples. Aligned characters might be referenced in an example, but the <ALIGN_CHAR> and <ENDALIGN_CHAR> tags must occur outside the example.

required terminator

<ENDALIGN_CHAR>

DESCRIPTION

The <ALIGN_CHAR> tag identifies the character that is to be used to signify a space in a table row. The tag translator replaces each occurrence of the character with a space. The space is the same width as the numeric characters in the font that is active. (All numeric characters of a font have a uniform width.)

Items in a column are aligned on the left by default. You may want to display a column of numbers aligned on the right. You can use the <ALIGN_CHAR> tag to define "#," for example, as the alignment character. You make all the numeric entries of equal length by prefixing the shorter entries with the alignment character. Because the numeric characters are of uniform width, they will then align on the right. You can also use the alignment character to

add space within the number. For example, you could separate the digits of a number at the thousands position.

EXAMPLES

1
```
<ALIGN_CHAR>(#)
<TABLE>
<TABLE_SETUP>(2\16)
<TABLE_ROW>(0#123#456#781\01234567.89)
<TABLE_ROW>(#######35#279\#######6.0)
<TABLE_ROW>(########4#341\####1429.857)
<ENDTABLE>
<ENDALIGN_CHAR>
```

This example defines the number sign as the alignment character. The number sign is then used to align the columns in a two-column list. This example may produce the following output:

0 123 456 789	01234567.89
35 279	6.0
4 361	1429.857

2
```
<TABLE>
<TABLE_SETUP>(2\16)
<TABLE_ROW>(0123456789\01234567.89)
<TABLE_ROW>(35279\6.0)
<TABLE_ROW>(4361\1429.857)
<ENDTABLE>
```

This example shows the two-column list used in the previous example without the use of an alignment character. This example may produce the following output:

0123456789	01234567.89
35279	6.0
4361	1429.857

3
```
<P>The sum may be padded on the left with blanks, resulting in
<ALIGN_CHAR>(#)
<KEEP>('###7.56')
<ENDALIGN_CHAR>
```

This example shows how blank output characters can be represented using the <ALIGN_CHAR> tag. This example may produce the output:

The sum may be padded on the left with blanks, resulting in '   7.56'

4
```
<P>The sum may be padded on the left with blanks, resulting in
<ALIGN_CHAR>(#\DELTA)
<KEEP>('###7.56')
<ENDALIGN_CHAR>
```

This example shows how delta output characters can be represented using the <ALIGN_CHAR> tag. This example may produce the output:

The sum may be padded on the left with delta characters, resulting in '∆∆∆7.56'

<ALIGN_NUMBER>

Specifies a numeric value with alignment characters.

FORMAT

<ALIGN_NUMBER> *(number)*

ARGUMENTS

number
Specifies a number, which can include commas and decimal points, that is to be aligned with other numbers in the same column. Fields in the number that are to be left blank may contain pound sign (#) characters representing blank numbers or semicolon (;) characters representing commas or decimal points.

related tags

- <ALIGN_AFTER>
- <ALIGN_CHAR>
- <TABLE_ROW>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <ALIGN_NUMBER> tag lets you align columns of numbers in a table, when the numbers you need to enter have suppressed leading or trailing fields. You enter the number using the pound-sign and semicolon characters to represent fields in the number that are to be blank on output.

Items in a column are aligned on the left by default. You may want to display a column of numbers aligned on the right, or aligned around the decimal point. To do this, you use the <ALIGN_NUMBER> tag to tag each number.

EXAMPLES

1

```
<table>
<table_setup>(2\8)
<table_row>(one\<align_number>(100,000,000.##))
<table_row>(one\<align_number>(###;###;240.40))
<table_row>(one\<align_number>(###;###;###.60))
<table_row>(one\<align_number>(###;230,425))
<endtable>
```

This example shows a series of numbers aligned at the decimal point. This example may produce the output:

one	100,000,000.
one	240.40
one	.60
one	230,425

2

```
<table>
<table_setup>(3\15\10)
<table_heads>(Item\Net Revenue\Percentage)
<table_row>(Fish knives\<align_number>(#;#20;435;##)\<align_number>(%#10.44))
<table_row>(Clam cleaners\<align_number>(1,432,064.23)\<align_number>(%#45.0#))
<table_row>(Shrimp peelers\<align_number>(#;###;245;##)\<align_number>(%###.86))
<endtable>
```

This table may produce the output:

Item	Net Revenue	Percentage
Fish knives	20,435	% 10.44
Clam cleaners	1,432,064.23	% 45.0
Shrimp peelers	245	% .86

<AMPERSAND>

Supplies an ampersand within an argument to a tag or in math.

FORMAT

<AMPERSAND>

ARGUMENTS

None.

related tags

- The following tags label other characters that must be tagged when they occur in an argument to a global tag:

  <BACKSLASH>
  <CPAREN>
  <OPAREN>
  <VBAR>

restrictions

Can only be used within an argument to a tag.

required terminator

None.

DESCRIPTION

Use the <AMPERSAND> tag to output an ampersand within an argument to a tag. Outside of an argument to an SDML tag, use the regular ampersand character rather than the tag.

The ampersand has a special meaning to the tag translator when it occurs within an argument to a tag. If you use a literal ampersand within an argument, it may be misinterpreted, and may cause an error in your output. Use the <AMPERSAND> tag to avoid any misinterpretation.

EXAMPLE

```
<SUBHEAD1>(Continuing the Line with <AMPERSAND> )
<P>Your BASIC statement may be formatted over two or more lines
by terminating all lines but the last with an ampersand (&).
```

This example can produce the following output.

Continuing the Line with &

Your BASIC statement may be formatted over two or more lines by terminating all lines but the last with an ampersand (&).

<APPENDIX>

Begins an appendix.

FORMAT

<APPENDIX> *(appendix-title[\ symbol-name])*

ARGUMENTS

appendix-title

Specifies the title for the appendix.

symbol-name

Specifies the symbol-name to be used in all cross-references to this appendix.

The value of the symbol in the cross-reference file is the number assigned to the appendix. If the appendix is an element in a book, the appendix number is determined based on the position of the appendix with respect to other appendixes in the book.

If the file being processed is not currently part of a book, the first appendix in the source file will be lettered A, and additional appendixes will be sequentially lettered thereafter.

If the file containing this tag is to be included in a bookbuild, the symbol-name is required.

related tags

- <SET_APPENDIX_LETTER>

restrictions

None.

required terminator

<ENDAPPENDIX>

DESCRIPTION

The <APPENDIX> tag starts the appendix section of a book and can contain any number of other tags. Appendixes contain supplementary material at the end of a book.

<BACKSLASH>

Supplies a backslash within an argument to a tag.

FORMAT

<BACKSLASH>

ARGUMENTS

None.

related tags

- The following tags label other characters that must be tagged when they occur in an argument to a global tag:

  <AMPERSAND>
  <CPAREN>
  <OPAREN>
  <VBAR>

restrictions

Can be used only within an argument to a tag.

required terminator

None.

DESCRIPTION

Use the <BACKSLASH> tag to output a backslash within an argument to a tag. Outside of an argument to an SDML tag use the regular backslash character rather than the tag.

Because the literal backslash character is used to separate the arguments in the argument list, you must use the <BACKSLASH> tag to include a backslash as part of your argument.

EXAMPLE

```
<SUBHEAD1>(Using the Backslash (<BACKSLASH>))
<P>Use the backslash character (\) to separate BASIC statements
on the same line.
```

This example may produce the following output:

Using the Backslash (\)

Use the backslash character (\) to separate BASIC statements on the same line.

As shown in this example, a simple backslash character is used in ordinary text. When a literal backslash is used in the argument to the <SUBHEAD1> tag, the text following the backslash is not output.

<BOX>

Produces a box that surrounds a user-specified character string.

FORMAT

<BOX> *(label)*

ARGUMENTS

label

Specifies the character string to be surrounded by the box.

related tags

None.

restrictions

Valid in any context except math.

The character string must be 15 characters or fewer. The box format might not be available for all output devices.

required terminator

None.

DESCRIPTION

The <BOX> tag produces a box that surrounds a user-specified character string in the printed output.

EXAMPLE

```
<P>In EDT, the <BOX>(GOLD) key
changes the function of the other keys on the keypad.
```

This example produces the following output:

In EDT, the [GOLD] key changes the function of the other keys on the keypad.

<CALLOUT>

Labels a callout in an example. The <CALLOUT> tag is identical to the <CO> tag.

FORMAT

<CALLOUT> *[(number)]*

ARGUMENTS

number

Specifies the number to be used as the callout. If not specified, the next number in the callout sequence is output.

related tags

- <CALLOUTS>
- <CALLOUT_REF>
- <CO>
- <LIST> (CALLOUT)

restrictions

None.

required terminator

None.

DESCRIPTION

The <CALLOUT> tag identifies a number that will be printed as a white letter in a black box in an example. It is used within the <CALLOUTS> and <ENDCALLOUTS> tags.

EXAMPLES

See the examples in the discussion of the <CALLOUTS> tag.

<CALLOUT_REF>

Labels a reference to a callout in text.

FORMAT

<CALLOUT_REF> *(callout number)*

ARGUMENTS

callout number
Specifies the number to be included in the callout.

related tags

- <CALLOUTS>
- <CALLOUT>
- <CO>
- The following tags label other types of references:

  <REFERENCE> (symbol)
  <GREF> (reference text)

restrictions

Invalid in the context of math.

Cannot be used in examples.

required terminator

None.

DESCRIPTION

The <CALLOUT_REF> tag identifies a reference in text to a callout number created by the <CALLOUT> tag in an example or figure.

EXAMPLE

```
<P>In <REFERENCE>(page_transitions_fig),
callout <CALLOUT_REF>(1)
labels the beginning of the process. The initial conditions are the same
as in the previous figure and the copy-on-reference bit is set.
```

This example may produce the following output:

In Figure 3-3, callout ❶ labels the beginning of the process. The initial conditions are the same as in the previous figure and the copy-on-reference bit is set.

<CALLOUTS>

Labels the beginning of a series of callouts contained in an example and enables the use of the <CALLOUT> and <CO> tags. (<CALLOUT> and <CO> are identical in their use.)

FORMAT

<CALLOUTS> *[([callout-number] \ [PREFIX])]*

ARGUMENTS

callout-number

Sets the number with which to begin numbering the callouts. If no number is specified, the numbering begins with one.

PREFIX

Determines that the callouts will appear before the line they label, rather than at the end of the line. All callouts in any one example must precede the line they label or appear inside or at the end of the line they label. Location of the callouts cannot be mixed.

related tags

- <CALLOUT>
- <CO>
- <CALLOUT_REF>
- <LIST> (CALLOUT)

restrictions

The <CALLOUTS> and <ENDCALLOUTS> tags must surround the examples within which the callouts are used.

The PREFIX argument, if specified, must be the second argument. The first argument can be null. For example, <CALLOUTS> (\PREFIX) is valid.

required terminator

<ENDCALLOUTS>

EXAMPLES

1

```
<CALLOUTS>
<CODE_EXAMPLE>
          DO WHILE (^EOF);  <CALLOUT>
             READ FILE(INFILE) INTO(Y);
             PUT LIST(Y);   <CALLOUT>
             END;
<ENDCODE_EXAMPLE>
<ENDCALLOUTS>
<LIST>(CALLOUT)
<LE>This is a <KEYWORD>(DO) statement.
<LE>This <KEYWORD>(PUT) statement outputs the line read.
<ENDLIST>
```

This example may produce the following output:

```
DO WHILE (^EOF);  ❶
   READ FILE(INFILE) INTO(Y);
   PUT LIST(Y);   ❷
   END;
```

❶ This is a **DO** statement.

❷ This **PUT** statement outputs the line read.

2

```
<CALLOUTS>(3\PREFIX)
<CODE_EXAMPLE>
                  <CALLOUT>    DO WHILE (^EOF);
                                       READ FILE(INFILE) INTO(Y);
                  <CALLOUT>       PUT LIST(Y);
                                       END;
<ENDCODE_EXAMPLE>
<ENDCALLOUTS>
```

This example shows how you can set the number from which to begin numbering the callout sequence and how you can cause the callouts to precede the line to which they belong by using the argument PREFIX. This example may produce the following output:

```
❸      DO WHILE (^EOF);
           READ FILE(INFILE) INTO(Y);
❹         PUT LIST(Y);
           END;
```

3

```
<CALLOUTS>
<CODE_EXAMPLE>
                  DO WHILE <CALLOUT>(9) (^EOF);
                     READ FILE(INFILE) INTO(Y);
                     PUT <CALLOUT>(12) LIST(Y);
                     END;
<ENDCODE_EXAMPLE>
<ENDCALLOUTS>
```

This example shows how you can place callouts inside of lines and how you can use an argument to the <CALLOUT> tag to control the number of the callout. This example may produce the following output:

```
DO WHILE ❾ (^EOF);
   READ FILE(INFILE) INTO(Y);
   PUT ⓬ LIST(Y);
   END;
```

<CENTER_LINE>

Specifies a line of text that is to be centered within the current margin.

FORMAT

<CENTER_LINE> *(text [\ { BIGSKIP | SMALLSKIP }])*

ARGUMENTS

text
Specifies a line of text to be centered.

BIGSKIP
SMALLSKIP
Specifies a set amount of vertical space to precede the element identified as a line or block of text. The actual amount of space created is determined by the document's design.

related tags

- <LINE>
- <RIGHT_LINE>

restrictions

Invalid in monospaced examples, in math, and in arguments to tags that provide title or heading text.

Centered text must fit within the current text margin. If you specify text that is too wide, the text formatter issues a warning message, and you should examine your output.

DESCRIPTION

The <CENTER_LINE> tag specifies a line of text that is to be centered within the current margin.

EXAMPLE

```
<P>Please include the following information:

<CENTER_LINE>(Name\smallskip)
<CENTER_LINE>(Address)
<CENTER_LINE>(Phone Number)
```

This example may produce output like the following:

Please include the following information:

Name

Address
Phone Number

<CHAPTER>

Begins a chapter.

FORMAT

<CHAPTER> *(chapter-title [\ symbol-name])*

ARGUMENTS

chapter-title

Specifies the name of the chapter. The book design may also use this argument in the running title.

symbol-name

Specifies the symbol-name to be used in all cross-references to this chapter.

The XREF file entry for the chapter will consist of the following:

- The value of the symbol is the number assigned to the chapter. If the chapter is an element in a book, the chapter number is determined based on the position of the chapter with respect to other chapters in the book.
- If the file being processed is not currently part of a book, the first chapter in the source file will be numbered 1, and additional chapters will be sequentially numbered thereafter.

Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores. Do not begin a symbol-name with an underscore.

related tags

- <REFERENCE>

restrictions

Invalid within <FRONT_MATTER> and <ENDFRONT_MATTER> .

If the file containing this tag is to be included in a bookbuild, the symbol-name is required.

required terminator

None.

DESCRIPTION

The <CHAPTER> tag labels the beginning of a chapter. When printed, the chapter title and number is printed in a larger font type. VAX DOCUMENT automatically increments chapter numbers.

EXAMPLE

To see an example of the output that a <CHAPTER> tag produces, refer to the beginning of this chapter.

<CHEAD>

Marks an unnumbered centered heading. Similar to the <CENTER_LINE> tag.

FORMAT

<CHEAD> *(heading-text)*

ARGUMENTS

heading-text
Specifies the text of the subsidiary heading.

related tags

- <CENTER>
- <HEAD1> through <HEAD6>
- <SUBHEAD1>
- <SUBHEAD2>

restrictions

None.

required terminator

None.

DESCRIPTION

The <CHEAD> tag labels an unnumbered subsidiary heading.

Subheadings are not numbered and do not appear in the table of contents. They cannot be readily used for cross-references and should be used only when the clarity of your exposition absolutely requires such a fine level of distinction.

EXAMPLE

```
<CHEAD>(How to Use the <TAG>(chead) Tag)
<P>The use of centered headings should be restricted to occasions when the
clarity of your exposition absolutely requires one.
```

This example might produce output like the following:

How to Use the <CHEAD> Tag

The use of centered headings should be restricted to occasions when the clarity of your exposition absolutely requires one.

<CHECK_FOR_INCLUSION>

Marks a file to ensure the file is included only once in the output.

FORMAT

<CHECK_FOR_INCLUSION> *(file-label)*

ARGUMENTS

file-label

Specifies a string that uniquely identifies this file. This string is not the name of the file, rather, it simply sets an internal switch to indicate that the following input should be read only once. The string can contain up to 15 alphanumeric characters or underscores and must not begin with an underscore character.

related tags

None.

restrictions

This tag is conditional, and has no relationship to the structure or content of the document.

required terminator

<ENDCHECK_FOR_INCLUSION>

DESCRIPTION

The <CHECK_FOR_INCLUSION> tag checks if the file containing the tag (the current input file) has been previously read. If it has, the rest of the file is ignored until the <ENDCHECK_FOR_INCLUSION> tag is found. If this is the first time that the file is being read, processing continues.

An input file that contains this tag will not be processed from the point that the text processor finds this tag until the terminating tag is found. This can be useful for specifying the name of a definitions file that has already been processed for an entire book and does not need to be processed for an individual element of the book.

You must use the <ENDCHECK_FOR_INCLUSION> tag at the bottom of a file in which the <CHECK_FOR_INCLUSION> appears. If you do not include the terminating tag, you might not get output after the <CHECK_FOR_INCLUSION> tag, or you may receive a fatal error from the tag translator when it does not find the specified label.

The position of the <CHECK_FOR_INCLUSION> tag within a file is important. Make sure to put the tag at the top of your file, to avoid partial processing of the file. Partial processing could result in illogical nesting of tags or other error conditions.

EXAMPLE

```
<CHECK_FOR_INCLUSION>(MY_DEFS)
   .
   .
   .
<ENDCHECK_FOR_INCLUSION>
```

This example shows how the <CHECK_FOR_INCLUSION> tag can be used to exclude a file (in this example, the file "MY_DEFS") from processing during a bookbuild. This can be useful if the file specified contains tag definitions that are defined within each file as well as in the local definition file. To force the tag translator to process the definitions twice would be a waste of processing time. Instead, it is better to identify the definitions file with this tag and exclude the file from processing.

<CO>

Labels a callout in an example. The <CO> tag is identical to the <CALLOUT> tag.

FORMAT

<CO> *[(number)]*

ARGUMENTS

number

Specifies the number to be used as the callout.

related tags

- <CALLOUT>
- <CALLOUTS>
- <LIST> (CALLOUT)

restrictions

None.

required terminator

None.

DESCRIPTION

The <CO> tag identifies a number in an example and is used within the <CALLOUTS> and <ENDCALLOUTS> tags.

EXAMPLES

See the examples in the discussion of the <CALLOUTS> tag.

<CODE_EXAMPLE>

Begins an example of code. Code consists of words or lines of instructions written in a programming language or a command language.

FORMAT

<CODE_EXAMPLE> *(code)*

or

<CODE_EXAMPLE> *[([KEEP] [WIDE[\ MAXIMUM]])]*
code_example text

.
.
.

<ENDCODE_EXAMPLE>

ARGUMENTS

code

Specifies a code fragment you want to insert into your text.

If this argument is not specified, the terminator <ENDCODE_EXAMPLE> is required.

KEEP

Specifies that the example is not to be broken across pages, that is, if the example does not fit on the current page, it will be placed on the next page. If the example itself does not fit on a single page of output, it will be broken anyway.

WIDE

Specifies that the width of the example exceeds the document's default width for text. Depending on the document type, this argument can be interpreted as follows:

- If the document style contains a left margin area that is normally used for headings, the example's width will span that area as well as the normal text area.
- If the document uses a multicolumn format, the example suspends the multicolumn output while the example is processed. The example will be output and multicolumn output is then restored.
- If the document style provides a range of sizes and styles for examples, this argument may be interpreted to mean that a specific size should be used for the example.

MAXIMUM

Can be used in conjunction with WIDE to indicate that the example may require additional adjustment to fit within the bounds of the text page. This argument must be used with discretion, and may not be suitable in all document styles.

related tags

- <INTERACTIVE>
- <VALID_BREAK>

restrictions

Indexing tags (<X> and <Y> tags) are not permitted within code examples.

Tab characters cannot be used to format code examples. You must use spaces rather than tabs.

Do not use text element tags within a code_example (for example, <P> , <LIST> , or <NOTE> .)

required terminator

<ENDCODE_EXAMPLE> —Required if the code example is not passed as an argument.

DESCRIPTION

The code that is begun with a <CODE_EXAMPLE> tag is distinguished typographically in the output. The size of the example, whether it will be indented, and how much it will be indented from the current left margin of text is controlled by the document design.

There are two types of code examples. The first is a short fragment that you want run in with the surrounding text. The second is one or more lines that you want broken out of the text that surrounds it (whether or not it is a "formal" example with its own number, caption, and entry in the table of contents).

In the first instance, where you want the example run in with the surrounding text, you pass the code as an argument to the <CODE_EXAMPLE> tag. In this instance, you do not need a terminating tag.

In the second instance, you should enter the code between the <CODE_EXAMPLE> and <ENDCODE_EXAMPLE> tags. In this context, the character spaces and blank lines you enter to format the code will be retained. Also, within this context you can use the <ELLIPSIS> tag to achieve a vertical ellipsis showing that you have omitted some lines of code. If your code example is longer than a few lines, use the <VALID_BREAK> tag to indicate the acceptable points for a page break.

EXAMPLES

1

```
<P>The <CODE_EXAMPLE>(WHILE INLOOP) statement causes the
following block to be repeated until the <VARIABLE>(INLOOP) variable
is set to FALSE.
```

This example illustrates the case of a short code example that is run in with the surrounding text. (One other related tag is also used.) The example may produce the following output:

The *WHILE INLOOP* statement causes the following block to be repeated until the *INLOOP* variable is set to FALSE.

2

```
<P>The call frame is built on the stack by the following four instructions:
<CODE_EXAMPLE>
     PUSHAB    B^SRVEXIT
     PUSHL     FP
     PUSHL     AP
     CLRQ      -(SP)
<ENDCODE_EXAMPLE>
```

This example illustrates the use of the <CODE_EXAMPLE> tag with a longer example that is broken out of the surrounding text. This example may produce the following output:

The call frame is built on the stack by the following four instructions:

```
PUSHAB    B^SRVEXIT
PUSHL     FP
PUSHL     AP
CLRQ      -(SP)
```

3

```
<P>The instruction sequence listed here (patterned after code in
module PROCSTRT) shows this second technique.
<CODE_EXAMPLE>
PUSHL     executive-mode-PSL
BSBB      DOREI
<ELLIPSIS>
PUSHL     user-mode-PSL
BSBB      DOREI
<ELLIPSIS>
DOREI:    REI
<ENDCODE_EXAMPLE>
```

This example shows a longer code example that uses the <ELLIPSIS> tag as well. This example may produce the following output:

The instruction sequence listed here (patterned after code in module PROCSTRT) shows this second technique.

```
PUSHL     executive-mode-PSL
BSBB      DOREI
   .
   .
   .
PUSHL     user-mode-PSL
BSBB      DOREI
   .
   .
   .
DOREI:    REI
```

<COMMENT>

Marks a portion of your SDML input file that you do not want to appear in your output. Text marked by a <COMMENT> tag is ignored by the tag translator during processing.

FORMAT

<COMMENT> *(comment-text)*

or

<COMMENT>
comment-text

.
.
.

<ENDCOMMENT>

ARGUMENTS

text
Specifies the text you do not want to appear in your output.

related tags

None.

restrictions

You cannot nest <COMMENT> and <ENDCOMMENT> tags.

required terminator

<ENDCOMMENT> — Required if the text is not passed as an argument.

DESCRIPTION

Comments can provide useful reminders for your own use or use by writers who may modify your SDML file in the future. Comments can also be used to exclude portions of text from the output file, but that you might wish to save for later inclusion. The text remains in the SDML file, but does not appear in the output.

EXAMPLE

```
<CODE_EXAMPLE>
        ;
        ; SECONDARY POOL COMMAND BUFFER BLOCKS
        ;
        .=0
000000  C.CLK:    .BLKW  1 ;LINK WORD
000002  C.CTCB:   .BLKW  1 ;TCB ADDRESS OF TASK TO RECEIVE COMMAND
000004  C.CUCB:   .BLKW  1 ;UCB ADDRESS IF RESPONSIBLE TERMINAL
000006  C.CCT:    .BLKW  1 ;CHARACTER COUNT, EXCLUDING TRAILING CR
000010  C.CSTS:   .BLKW  1 ;STATUS MASK
<COMMENT>
000012  C.CMCD:            ;SYSTEM MESSAGE CODE
000012  C.CSO:    .BLKW  1 ;STARTING OFFSET OF VALID COMMAND TEXT
<ENDCOMMENT>
000014  C.CTR:    .BLKW  1 ;TERMINATING CHARACTER
000015  C.BLK:    .BLKW  1 ;SIZE OF PACKET IN SEC POOL (32 WD.) BLOCKS
000016  C.CTXT:            ;COMMAND TEXT, FOLLOWED BY CR
<ENDCODE_EXAMPLE>
```

This example shows a code example with comments embedded in the code. The tag translator ignores the text between the <COMMENT> tag and the <ENDCOMMENT> tag. This example may produce the following output:

```
        ;
        ; SECONDARY POOL COMMAND BUFFER BLOCKS
        ;
        .=0
000000  C.CLK:    .BLKW  1 ;LINK WORD
000002  C.CTCB:   .BLKW  1 ;TCB ADDRESS OF TASK TO RECEIVE COMMAND
000004  C.CUCB:   .BLKW  1 ;UCB ADDRESS IF RESPONSIBLE TERMINAL
000006  C.CCT:    .BLKW  1 ;CHARACTER COUNT, EXCLUDING TRAILING CR
000010  C.CSTS:   .BLKW  1 ;STATUS MASK

000014  C.CTR:    .BLKW  1 ;TERMINATING CHARACTER
000015  C.BLK:    .BLKW  1 ;SIZE OF PACKET IN SEC POOL (32 WD.) BLOCKS
000016  C.CTXT:            ;COMMAND TEXT, FOLLOWED BY CR
```

<CONDITION>

Marks a section of an SDML file that is not processed unless one of the arguments to the tag matches the argument in the related <SET_CONDITION> tag.

FORMAT

<CONDITION> *(condition-name [\ condition-name] [\ condition-name])*

ARGUMENTS

condition-name

Specifies a condition name created with the <SET_CONDITION> tag.

related tags

- <SET_CONDITION>

restrictions

<CONDITION> tags cannot be nested. There must be a strict correspondence between <CONDITION> and <ENDCONDITION> tags.

In monospaced examples, <CONDITION> and <ENDCONDITION> tags cannot occupy the same line as text for the example.

required terminator

<ENDCONDITION> — Required if the text is not passed as an argument.

DESCRIPTION

The <CONDITION> tag indicates that the following section of text should be processed only upon the condition that one of the tag's arguments matches the argument in the related <SET_CONDITION> tag. If the two tags' arguments are identical, the marked text is processed and included in the output file. If the arguments differ, the text is excluded from processing. The <ENDCONDITION> tag marks the end of the conditional text.

This tag can be very useful if you must produce two or more documents that differ in minor ways but contain much of the same text. It is efficient to maintain one SDML file and identify the portions of text that are unique to each document.

Before processing the SDML file, you can specify the condition-name for the desired document in the <SET_CONDITION> tag or in the /CONDITION qualifier in the DOCUMENT command line. When the tag translator processes the file, it includes all portions of text tagged with that condition-name argument and ignores all portions of text tagged with any other condition name. To obtain another version of the document, change the condition name in the argument to the <SET_CONDITION> tag, and reprocess the file.

When you supply more than one condition-name argument to the <CONDITION> tag, only one of the condition names need be present for the conditional text to be processed.

This is shown in the first example.

EXAMPLES

1

```
<CONDITION>(Christmas)
<P>Christmas, by convention, is celebrated on December 25th . . .
<ENDCONDITION>

<CONDITION>(Chanukah)
<P>Chanukah is called the Festival of Lights . . .
<ENDCONDITION>

<CONDITION>(Wash_Bday)
<P>Washington's Birthday is on February 22nd . . .
<ENDCONDITION>

<CONDITION>(Christmas\Chanukah)
<HEAD1>(Religious Holidays)
<P>This paragragh contains general information about several religious
holidays . . .
<ENDCONDITION>
```

This example shows how a file could be organized if you were writing about holidays in general, and Christmas, Chanukah, and Washington's Birthday in particular. Any of three different condition states could be set at the top with the tag <SET_CONDITION> (Christmas), <SET_CONDITION> (Chanukah), or <SET_CONDITION> (Wash_Bday). The example contains four paragraphs, with each paragraph conditionalized differently.

If the file is processed with <SET_CONDITION> (Christmas), the information on Christmas and religious holidays is processed. If the file is processed with <SET_CONDITION> (Chanukah), the information on Chanukah and religious holidays is processed. If the file is processed with <SET_CONDITION> (Wash_Bday), the information on Washington's birthday is processed.

The function of the <SET_CONDITION> tag can be obtained by using the /CONDITION qualifier on the DOCUMENT command line instead.

2

```
<SET_CONDITION>(VMS)
    .
    .
    .
When the
<CONDITION>(VMS)VAX/VMS<ENDCONDITION>
<CONDITION>(RSX)RSX11M/RSX11M-PLUS<ENDCONDITION>
command language interpreter translates the logical name . . .
```

This example shows how one of two clauses can be omitted from processing based on the condition name specified in the <SET_CONDITION> tag. The section of text that is identified with <CONDITION> (VMS) would be processed and placed in the output file, whereas the other text would be omitted. This example would have the following output:

When the VAX/VMS command language interpreter translates the logical name . . .

3

```
<SET_CONDITION>(RSX)
<P>When RSX . . .
   .
   .
   .
<CONDITION>(VMS\RSX)
<P>When the operating system . . .
   .
   .
   .
<ENDCONDITION>
<CONDITION>(RSTS)
<P>When RSTS . . .
   .
   .
   .
<ENDCONDITION>
```

This example shows how text that is applicable to the two condition names "VMS" and "RSX" can be differentiated from text that is applicable to the condition-name "RSTS." By supplying both the VMS and RSX arguments to the first <CONDITION> tag, the writer ensures that the text will be included in the output if either the VMS or RSX condition-name is set. (Other text in the same SDML file might be conditional for only one of these condition-names.)

<CONTENTS_FILE>

Specifies that the table of contents output file for a document should be included when the document is processed.

FORMAT

<CONTENTS_FILE> *[(file-spec)]*

ARGUMENTS

file-spec

If you place the <CONTENTS_FILE> tag in an SDML file that will be included in another file later, you must specify the exact file name of the contents as the <CONTENTS_FILE> tag's argument.

restrictions

<CONTENTS_FILE> is valid only within the context of front matter or within a profile of a book.

related tags

- <PROFILE>

required terminator

None.

DESCRIPTION

The <CONTENTS_FILE> tag specifies the position in a source file where a table of contents output file should be included. This tag does not produce a table of contents, but simply indicates placement of the contents file. A contents file is produced when the qualifier /CONTENTS is specified on the DOCUMENT command line.

The profile tags <CONTENTS_FILE> and <INDEX_FILE> can be placed in either the profile of a book or in a source file. If you are creating a book (with a profile to be processed through a bookbuild), place these tags in the profile to be sure of correct placement in the output.

A contents file always receives the filetype .DVI *_device,* where *device* is the type of output device you specified on the command line. For more information on contents generation, see Chapter 7.

To create a table of contents from an individual file that contains a <CONTENTS_FILE> tag, specify the /CONTENTS qualifier on the command line.

EXAMPLE

To see the result of the <CONTENTS_FILE> tag, refer to the table of contents in this manual. The <CONTENTS_FILE> tag was placed in the profile file before the preface file.

<COPYRIGHT_DATE>

Inserts a copyright date line on the copyright page along with other system-specific copyright information.

FORMAT

<COPYRIGHT_DATE> *(date[\ owner])*

ARGUMENTS

date

Specifies official printing date information for the book.

owner

Specifies the owner of the copyright. Local conventions might preclude use of this argument.

restrictions

The <COPYRIGHT_DATE> tag is valid only within the context of a <COPYRIGHT_PAGE> ... <ENDCOPYRIGHT_PAGE> .

required terminator

None.

DESCRIPTION

The <COPYRIGHT_DATE> inserts a copyright date line on the copyright page along with other system-specific copyright information.

EXAMPLE

```
<Copyright_page>
<Copyright_date>(1987\Tomato Magnates, Inc.)
<Endcopyright_page>
```

This example produces a full page of output with the following text:

Copyright ©1987 Tomato Magnates, Inc.

<COPYRIGHT_PAGE>

Begins a copyright page and enables copyright page tags.

FORMAT

<COPYRIGHT_PAGE>

ARGUMENTS

None.

restrictions

This tag can only be used in the context of a copyright page and in the front matter of a book.

required terminator

<ENDCOPYRIGHT_PAGE>

related tags

- <FRONT_MATTER>
- <PRINT_DATE>
- <COPYRIGHT_DATE>

DESCRIPTION

The following copyright page tags that are enabled by <COPYRIGHT_PAGE> :

- <PRINT_DATE>
- <COPYRIGHT_DATE>

EXAMPLE

```
<FRONT_MATTER>(front)
<COPYRIGHT_PAGE>
<PRINT_DATE>(March 1987)
<COPYRIGHT_DATE>(1987)
<ENDCOPYRIGHT_PAGE>
<ENDFRONT_MATTER>
```

This example shows the order in which the copyright page tags are used. Notice that you must enable the copyright page tags within the <FRONT_MATTER> tag. The output of this example is a separate copyright page, containing the print and copyright dates, in the front matter of the book.

To see an example of all the front matter tags in their correct order, refer to the example in the <FRONT_MATTER> tag.

<CP>

Marks the continuation of a paragraph that has been interrupted by another text element.

FORMAT <CP>

related tags

- <P>

ARGUMENTS

None.

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <P> and <CP> tags may have no visible difference in effect. In a book design in which all new paragraphs begin flush left, the formatted results achieved by a <CP> tag and by a <P> tag are identical. However, in a book design in which paragraphs are indented, the continued paragraph may still begin flush left.

In some instances, the <CP> tag will keep text that follows a list or monspaced example from being "widowed" at the top of the following page. In other words, a continued paragraph is more closely attached to the text element it follows.

EXAMPLE

```
<P>Each time you log in, the system automatically executes
two types of login command procedures:
<LIST>(UNNUMBERED)
<LE>A system login command procedure
<LE>Your personal login command procedure
<ENDLIST>
<CP>These login procedures are described in the following sections.
```

This example may produce the following output using a book design in which paragraphs are indented:

Each time you log in, the system automatically executes two types of login command procedures:

- A system login command procedure
- Your personal login command procedure

These login procedures are described in the following sections.

<CPAREN>

Supplies an unmatched closing parenthesis in an argument to a tag.

FORMAT

<CPAREN>

ARGUMENTS

None.

related tags

- The following tags label other characters that must be tagged when they occur in an argument to a global tag:

  <AMPERSAND>
  <BACKSLASH>
  <OPAREN>
  <VBAR>

restrictions

None.

required terminator

None.

DESCRIPTION

The <CPAREN> tag can be used anywhere to insert a closed parenthesis character into text. However, it is only beneficial (in terms of keystrokes and control of the output) as an unmatched closing parenthesis in an argument passed to a tag.

An unmatched parenthesis in an argument can cause errors when processed because the parentheses are used to determine the beginning and ending of an argument list. The <CPAREN> tag inserts the closed parenthesis character but is not evaluated as a closed parenthesis.

EXAMPLE

```
<SUBHEAD1>(Using a Closed Parenthesis
<PARENDCHAR>(<CPAREN>) in an Argument to a Tag)
```

This example may produce the following output:

Using a Closed Parenthesis ( )) in an Argument to a Tag

<DATE>

Produces the current system date or time, or the user-specified date and time.

FORMAT

<DATE> *[(FULL*
date-text)]

ARGUMENTS

FULL

Produces a full VMS date and time string in the format "dd-mmm-yyyy hh:mm:ss.hh." If no argument is specified, only the date may be produced in the format "Month day, year." (The precise format may be modified in different doctypes.)

date-text

Text that you provide as the current date. If no argument is specified, only the date may be produced in the format "Month day, year." (The precise format may be modified in different doctypes.)

related tags

None.

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <DATE> produces a date that is based either on the time at which the tag translator begins execution, or the time as specified by the user. The <DATE> tag produces a date in one of three forms as shown in the following table:

<DATE>	February 26, 1988
<DATE> (FULL)	26-FEB-1988 15:46:00.29
<DATE> (February 26th, 1988 A.D.)	February 26th, 1988 A.D.

Note that even if the <DATE> (FULL) tag is specified multiple times in a source file, it will always produce the same value and so cannot be used for timing information.

EXAMPLES

1 `<DATE>`

Example 1 may produce the following output: June 29, 1988

2 `<DATE>(FULL)`

Example 2 produces the following output: 29-JUN-1988 16:40:33.52

<DEFINE_BOOK_NAME>

Defines the title of a book and associates a user-defined symbol-name with that title for later reference.

FORMAT

<DEFINE_BOOK_NAME> *(symbol-name \ title)*

ARGUMENTS

symbol-name

Specifies the symbol that is associated with the title of the book. Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

title

Specifies the exact text of the book's title.

related tags

- <DEFINE_SYMBOL>
- <REFERENCE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <DEFINE_BOOK_NAME> tag specifies a book's title and adds an associated user-defined symbol-name to the cross-reference file. When you subsequently reference this title with the <REFERENCE> tag, supplying the same symbol-name as an argument to <REFERENCE> , the book title is retrieved from the cross-reference file and is substituted for the reference.

Depending on the document type, the title may be output with emphasized text (for example, italicized).

Note that the symbol-name argument is the *first* argument to the tag. In text element tags, the symbol-name is always the second argument. Only the <DEFINE_BOOK_NAME> and <DEFINE_SYMBOL> tags take a symbol-name as a first argument. Placing the symbol-name as the first argument to these tags makes it easy to keep track of your symbol-names when you list them in a local definitions file. All of the symbol-names will be aligned and easy to find in the file.

EXAMPLE

```
<DEFINE_BOOK_NAME>(games_book\Book of Games, Volume 2)
```

This example illustrates the use of the <DEFINE_BOOK_NAME> tag for defining the symbol-name of the book *"Book of Games"* as *games_book*. The tag <REFERENCE> (games_book) can be used anywhere in the document to refer to this book name.

<DEFINE_SYMBOL>

Associates a string of text with a user-defined symbol, so that the text can be referenced via this symbol throughout the document.

FORMAT

<DEFINE_SYMBOL> *(symbol-name \ text-string)*

ARGUMENTS

symbol-name

Specifies the name assigned to the symbol. All symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

text-string

Specifies the text that will be referenced via the symbol-name. Throughout the document, you can specify the symbol-name in the <REFERENCE> tag, and this text-string will be substituted for the symbol-name during processing.

related tags

- <DEFINE_BOOK_NAME>
- <DELAYED>
- <REFERENCE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <DEFINE_SYMBOL> tag specifies a string of text and associates the text with a user-defined symbol-name. It adds the symbol-name and the text to the symbol table. When you subsequently reference this text by using the <REFERENCE> tag and specifying the same symbol-name as its argument, the current value of the symbol-name is retrieved from the table and the text is substituted for the reference.

For more information on the use of the symbol table, see Chapter 6.

Note: The symbol-name argument to the <DEFINE_SYMBOL> tag must be on the same line as the tag; spaces and carriage returns are significant and are interpreted as part of the symbol-name.

Only the <DEFINE_SYMBOL> and <DEFINE_BOOK_NAME> tags use a symbol-name as a first argument. In text element tags, the symbol-name is often the second argument. Placing the symbol-name as the first argument to these tags makes it easy to keep track of your symbol-names when you list them in a local definitions file. All of the symbol-names will be aligned and easy to find in the file.

EXAMPLE

```
<define_symbol>(set_logical\RTL routine Set Logical Name, LIB$SET_LOGICAL,)

<P>The <REFERENCE>(set_logical) requests the calling process's CLI to
define or redefine a supervisor-mode logical name.
```

This example shows how a phrase, "RTL routine Set Logical Name, LIB$SET_LOGICAL," can be defined as a symbol-name, "set_logical," and then referenced in text by that symbol-name. By referencing the symbol-name in the places you want the phrase to occur, you can save keystrokes and ensure that you never have a typing error in that phrase.

The example produces the following output:

The RTL routine Set Logical Name, LIB$SET_LOGICAL, requests the calling process's CLI to define or redefine a supervisor-mode logical name.

<DEFINITION_LIST>

Begins a definition list.

FORMAT

<DEFINITION_LIST>

ARGUMENTS

None.

related tags

- <DEFINITION_LIST_HEAD>
- <DEFLIST_ITEM>
- <DEFLIST_DEF>

restrictions

None.

required terminator

<ENDDEFINITION_LIST>

DESCRIPTION

A definition list contains paired entries, consisting of the item being defined, introduced by the <DEFLIST_ITEM> tag; and the item's definition, introduced by the <DEFLIST_DEF> tag. The definition list differs from a list created with the <LIST> tag in that the definition list items are not numbered or called-out in any way.

The definition list can be given a heading with the <DEFINITION_LIST_HEAD> tag. Each list can contain one or more items, tagged with <DEFLIST_ITEM> , and each item can have a definition, tagged with <DEFLIST_DEF> .

EXAMPLE

```
<definition_list>
<deflist_item>(Hargreaves, James)
<deflist_def>
d. 1778.  English inventor.
<deflist_item>(Harris, Joel Chandler)
<deflist_def>
1848-1908.  American author.
<deflist_item>(Harvey, William)
<deflist_def>
1578-1657.  English physician and anatomist.
<deflist_item>(Hauptmann, Gerhard)
<deflist_def>
1862-1946.  German author.
<enddefinition_list>
```

This example may produce the following output:

Hargreaves, James
d. 1778. English inventor.

Harris, Joel Chandler
1848-1908. American author.

Harvey, William
1578-1657. English physician and anatomist.

Hauptmann, Gerhard
1862-1946. German author.

<DEFINITION_LIST_HEAD>

Supplies the heading to precede a definition list. Output formatting is controlled by the document-type.

FORMAT

<DEFINITION_LIST_HEAD> *(heading-text)*

ARGUMENTS

heading-text

Specifies the text for a heading to precede the definition list.

related tags

- <DEFINITION_LIST>
- <DEFLIST_ITEM>
- <DEFLIST_DEF>

restrictions

Can be used only in the context of a <DEFINITION_LIST> tag.

required terminator

None.

DESCRIPTION

The <DEFINITION_LIST_HEAD> tag adds a heading to a list of defined items specified by the <DEFINITION_LIST> tag.

EXAMPLE

```
<definition_list>
<definition_list_head>(Worldwide Associates)
<deflist_item>(IAAF)
<deflist_def>
International Amateur Athletic Federation.
<deflist_item>(IAEA)
<deflist_def>
International Atomic Energy Agency.
<enddefinition_list>
```

This example might have the following output, depending on the document type specified:

WORLDWIDE ASSOCIATES

IAAF

International Amateur Athletic Federation.

IAEA

International Atomic Energy Agency.

<DEFLIST_DEF>

Begins the text that defines an item in a definition list.

FORMAT

<DEFLIST_DEF> text

ARGUMENTS

text
Specifies the text of the definition.

related tags

- <DEFINITION_LIST>
- <DEFINITION_LIST_HEADS>
- <DEFLIST_ITEM>

restrictions

Can be used only in the context of a <DEFINITION_LIST> tag.

required terminator

None.

DESCRIPTION

A definition list contains paired entries, consisting of the item being defined, introduced by the <DEFLIST_ITEM> tag; and the item's definition, introduced by the <DEFLIST_DEF> tag.

EXAMPLE

See the example in the discussion of the <DEFINITION_LIST> tag.

<DEFLIST_ITEM>

Marks an item to be defined in a definition list.

FORMAT

<DEFLIST_ITEM> *(item[\ item...])*

ARGUMENTS

item
Specifies the item to be defined. Up to seven items can be stacked.

related tags

- <DEFINITION_LIST>
- <DEFINITION_LIST_HEADS>
- <DEFLIST_DEF>

restrictions

Can be used only in the context of a <DEFINITION_LIST> tag.

required terminator

<DEFLIST_DEF>

DESCRIPTION

A definition list contains paired entries, consisting of the item being defined, introduced by the <DEFLIST_ITEM> tag; and the item's definition, introduced by the <DEFLIST_DEF> tag.

EXAMPLE

```
<definition_list>
<deflist_item>(Hargreaves, James\d. 1778.)
<deflist_def>English inventor.
<deflist_item>(Harris, Joel Chandler\Born 1848.\Died 1908.)
<deflist_def>American author.
<enddefinition_list>
```

This example may have the following output:

Hargreaves, James
d. 1778.
English inventor.

Harris, Joel Chandler
Born 1848.
Died 1908.
American author.

<DELAYED>

Allows you to specify text that contains tags in an argument to another tag. Execution of tags is delayed until the text is output.

FORMAT

<DELAYED> *(delayed-text)*
or

<DELAYED>
delayed-text

.
.
.

<ENDDELAYED>

ARGUMENTS

delayed-text
The text containing tags whose execution is delayed until the text is output.

related tags

- <DEFINE_BOOK_NAME>
- <DEFINE_SYMBOL>

restrictions

The <DELAYED> tag may not be nested within itself.

required terminator

<ENDDELAYED> —Required if the delayed text is not specified as an argument.

DESCRIPTION

You can use a tag within an argument to another tag. This is called *nesting* a tag in an outer tag's argument. Normally, the tag translator evaluates the nested tag first, then evaluates the outer tag.

When the outer tag is the <DEFINE_SYMBOL> tag, a nested tag in the text argument will be evaluated before the <DEFINE_SYMBOL> tag is evaluated. Often, this leads to an undesirable result. You would like the text argument of the <DEFINE_SYMBOL> tag to be preserved as it is stated until the text is recalled at another place by the <REFERENCE> tag.

The <DELAYED> tag can be used to surround the text argument of a <DEFINE_SYMBOL> tag. The effect of the <DELAYED> tag is to delay the evaluation of the tags within it until those tags are about to be output where the symbol has been referenced.

EXAMPLE

```
<define_symbol>(TEMP_CHART_EX\<delayed>
<example>(Temperature Chart)
<code_example>
      Centigrade   0    Fahrenheit  32
      Centigrade 100    Fahrenheit 212
<endcode_example>
<endexample>
<enddelayed>)
```

The symbol TEMP_CHART_EX is associated with the text string that was enclosed by the <DELAYED> and <ENDDELAYED> tags. None of the enclosed tags will have been evaluated.

Subsequently, the symbol may be referenced by a <REFERENCE> tag as in the following code fragment.

```
<REFERENCE>(temp_chart_ex)
```

The previous code fragment may have the following output.

Example x—x Temperature Chart

```
Centigrade   0    Fahrenheit  32
Centigrade 100    Fahrenheit 212
```

<DOCTYPE>

Specifies the document type for your file. This tag is for informational purposes only.

FORMAT

<DOCTYPE> *(document-type)*

ARGUMENTS

document-type

Specifies the design you want to use for your file.

restrictions

None.

required terminator

None.

DESCRIPTION

The <DOCTYPE> tag specifies the document type for your file. The designated doctype determines how the file's formatted output will appear. Each doctype follows a book design as determined by the writer and editor.

You do not have to include the <DOCTYPE> tag in your file in order to process your file. You can include this tag for information purposes. You must specify the doctype in the DOCUMENT command line. Doing so allows you to process your file using different doctypes to determine which one suits your needs without having to modify the file.

Table 9–1 summarizes the supported doctypes and the DOCUMENT command keyword names to be specified in processing these doctypes.

Table 9–1 Supported Document Types

Document Type Keyword	Usage
ARTICLE	Articles
LETTER	Letters or memos
MANUAL	Users' manuals
MILSPEC	Military specifications
OVERHEADS	Overhead slides for transparencies
REPORT	General-purpose documents
SOFTWARE	User manuals containing software-specific information

In addition, a number of the document types listed in Table 9–1 have a choice of designs. The design selected is specified during document processing by appending the design name to the document-type keyword. Table 9–2 summarizes the key characteristics of the doctypes and their designs.

Table 9–2 Summary of Alternate Designs for Doctypes

Keywords	Size	Headings[1]	Characteristics
ARTICLE	$8\frac{1}{2} \times 11$	U	Two columns for text
LETTER	$8\frac{1}{2} \times 11$	U	Uses full page width
MANUAL.GUIDE	7×9	N	Chapter-oriented; headings unruled
MANUAL.PRIMER	7×9	U	Chapter-oriented with sequential page numbering; headings unruled
MANUAL.REFERENCE	$8\frac{1}{2} \times 11$	N	Chapter-oriented; headings unruled
MILSPEC	$8\frac{1}{2} \times 11$	N	Chapter-oriented with sequential page numbering; two-line running headings unruled
OVERHEADS	$8\frac{1}{2} \times 11$	—	—
OVERHEADS.35MM	$6\frac{1}{2} \times 5\frac{1}{2}$	—	—
REPORT	$8\frac{1}{2} \times 11$	N	Uses the full page width for text; headings are not ruled
REPORT.TWOCOL	$8\frac{1}{2} \times 11$	N	Two columns for text
SOFTWARE.BROCHURE	7×9	U	Headings set in left margin, ruled
SOFTWARE.GUIDE	7×9	N	Headings set in left margin, ruled
SOFTWARE_HANDBOOK	7×9	N	Headings set in left margin, ruled
SOFTWARE.POCKET_REFERENCE	$5\frac{1}{2} \times 7$	U	Headings optionally numbered or unnumbered
SOFTWARE.REFERENCE	$8\frac{1}{2} \times 11$	N	Headings set in left margin, ruled
SOFTWARE.SPECIFICATION	$8\frac{1}{2} \times 11$	N	Headings unruled; uses full page width

[1]N=numbered headings, U=unnumbered headings

Tags unique to specific document types are summarized in the *VAX DOCUMENT User Manual, Volume 2.*

EXAMPLE

```
<DOCTYPE>(report)
```

This example shows how a book's doctype can be specified as "report."

Refer to the document *VAX DOCUMENT Design Samples* for examples of the output of the various doctypes.

<DOUBLE_QUOTE>

Supplies a double quotation mark as it appears on the screen.

FORMAT

<DOUBLE_QUOTE>

ARGUMENTS

None.

related tags

- <SINGLE_QUOTE>
- <QUOTE>

restrictions

None.

required terminator

None.

DESCRIPTION

Typesetters distinguish between opening quotation marks and closing quotation marks, for example, "a quoted string." Your terminal, on the other hand, has two possible characters you can use for quotation marks: the double quote, (") (ASCII 34) and the single quote (') (ASCII 39). In programming languages, a quoted string is usually delimited at each end with the same character, either *"a quoted string,"* or *'a quoted string.'*

If you use the terminal's double quotation mark in normal text in your SDML file, you get the typesetter's closing quotation mark in the output. Thus, if you enter *"abc"* in your SDML file, you will see ″abc″ in your printed output. This is usually undesirable.

You have two choices:

- To cause your typeset output to show a double quotation mark as it is shown on the terminal, use the <DOUBLE_QUOTE> tag. This is your most likely choice for showing examples of user input or screen displays.
- To cause your typeset output to follow typesetting conventions by using distinct opening and closing quotation marks, use the <QUOTE> (text) tag. You would use this in citing a passage from another book.

Thus, there are very few occasions in which you press the double quote character on your keyboard while editing an SDML file.

EXAMPLES

1

```
<P>You can cause the translation of a symbol by using a double
quotation mark (<DOUBLE_QUOTE>) directly in front of it.
```

This example shows the use of the <DOUBLE_QUOTE> tag. It may produce the following output:

You can cause the translation of a symbol by using a double quotation mark (") directly in front of it.

2

```
<P>You can cause the translation of a symbol by using a double
quotation mark (") directly in front of it.
```

This example shows what your output would be like without using the tag, but just entering the character from the keyboard.

You can cause the translation of a symbol by using a double quotation mark (") directly in front of it.

<ELEMENT>

Identifies a file that contains an element of a book.

FORMAT

<ELEMENT> *(file-spec)*

ARGUMENTS

file-spec
Specifies a file. The file type must be supplied.

related tags

- <PROFILE>
- <INCLUDES_FILE>

restrictions

Must be used in the context of a profile.

required terminator

None.

DESCRIPTION

A profile of a book is required in order to build (process) a book. A book's profile contains an ordered list of the elements that comprise the book. In the profile, you label each of these elements with an <ELEMENT> tag. Only those files listed with <ELEMENT> tags are included in the book during the bookbuild. They should be listed in the profile in the order in which they are presented in the book.

A file specified as an element must begin with one of the following tags:

- <APPENDIX>
- <CHAPTER>
- <FRONT MATTER>
- <GLOSSARY>
- <PART>

For more information on creating a profile and bookbuilding, see Chapter 4.

EXAMPLE

Assume that a writer is writing a book that contains several chapters and that one of the chapters is the file My_Intro_Chap.SDML. At the top of this file is included the tag: <CHAPTER> (Introduction\intro_chap).

Along with the other <ELEMENT> tags for this book, the profile for this book would then contain the following:

```
<ELEMENT> (My_Intro_Chap.SDML)
```

<ELLIPSIS>

Supplies vertical ellipsis points to show omitted material in text and tables.

FORMAT

<ELLIPSIS>

ARGUMENTS

None.

related tags

- <HELLIPSIS>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <ELLIPSIS> tag supplies vertical ellipsis points to show omitted material in text and tables. The vertical ellipsis is positioned with respect to the current left margin.

EXAMPLE

```
<P>The instruction sequence listed here (patterned after code in
module PROCSTRT) shows this second technique.
<CODE_EXAMPLE>
PUSHL     executive-mode-PSL
BSBB      DOREI
<ELLIPSIS>
PUSHL     user-mode-PSL
BSBB      DOREI
<ELLIPSIS>
DOREI:    REI
<ENDCODE_EXAMPLE>
```

This example may produce the following output:

The instruction sequence listed here (patterned after code in module PROCSTRT) shows this second technique.

```
PUSHL     executive-mode-PSL
BSBB      DOREI
   .
   .
   .
PUSHL     user-mode-PSL
BSBB      DOREI
   .
   .
   .
DOREI:    REI
```

<EMPHASIS>

Marks a word or phrase for distinctive typographical treatment.

FORMAT

<EMPHASIS> *(text* [*\BOLD* | *\ITALIC* | *\SMALLCAPS* | *\SMALL_BOLDCAPS*] *)*

ARGUMENTS

TEXT
Specifies the text to be emphasized.

BOLD
Indicates that the text should be set in a boldface font, rather than the default italic.

ITALIC
Indicates that the text should be set in the current italic font. This is the default.

SMALLCAPS
Indicates that the text is to be set in small capital letters.

SMALL_BOLDCAPS
Indicates that the text is to be set in small capital letters.

related tags

- <QUOTE>
- <UNDERLINE>

restrictions

None.

required terminator

None.

DESCRIPTION

The default form of emphasis is to set the text in an italic font. Use one of the optional keyword arguments to change the emphasis to another font.

EXAMPLE

```
<P>An <EMPHASIS>(overuse) of the <TAG>(emphasis) tag
<EMPHASIS>(inevitably\bold) <EMPHASIS>(reduces\small_boldcaps) its
<EMPHASIS>(smallcaps)(effectiveness).
```

This example may produce the following output:

An *overuse* of the <EMPHASIS> tag **inevitably** **REDUCES** its EFFECTIVENESS.

<ENDCOPYRIGHT_PAGE>

Terminates the copyright page and optionally provides text that may be used on a document-type specific basis.

FORMAT

<ENDCOPYRIGHT_PAGE> *[(identification)]*

ARGUMENTS

identification

Is a local convention for information to be supplied on the copyright page.

restrictions

This tag can only be used in the context of a copyright page.

required terminator

None.

DESCRIPTION

The <ENDCOPYRIGHT_PAGE> terminates the copyright page and optionally provides text that may be used on a document-type specific basis.

EXAMPLE

```
<FRONT_MATTER>(front)
<COPYRIGHT_PAGE>
<PRINT_DATE>(March 1987)
<COPYRIGHT_DATE>(1987)
<ENDCOPYRIGHT_PAGE>
<ENDFRONT_MATTER>
```

This example shows the order in which the copyright page tags are used. Notice that you must enable the copyright page tags within the <FRONT_MATTER> tag. The output of this example is a separate copyright page in the front matter of the book, and the page contains the print and copyright dates.

To see an example of all the front matter tags in their correct order, refer to the example in the <FRONT_MATTER> tag.

<ENDPART_PAGE>

Terminates a part page and optionally specifies paging attributes for text that follows.

FORMAT

<ENDPART_PAGE> *[(RENUMBER)]*

ARGUMENTS

RENUMBER

Specifies that the part page is to be numbered 1. If this argument is not specified, page numbering continues following the part page with the next odd-numbered page.

restrictions

This tag must be used in the context of a part page.

required terminator

None.

DESCRIPTION

The <ENDPART_PAGE> tag terminates a part page and optionally specifies paging attributes for text that follows.

<ENDTITLE_PAGE>

Terminates a title page and optionally specifies text to appear at the bottom of the title page.

FORMAT

<ENDTITLE_PAGE> *(text)*

ARGUMENTS

text
Specifies a line to appear at the bottom of the title page.

restrictions

The <ENDTITLEPAGE> tag is valid only within the context of a title page.

required terminator

None.

DESCRIPTION

The <ENDTITLE_PAGE> tag terminates a title page and optionally specifies text to appear at the bottom of the title page.

EXAMPLE

Because the <ENDTITLE_PAGE> tag only terminates the title page, refer to the title page of this manual for an example of title page output.

<EXAMPLE>

Labels the beginning of a formal example.

FORMAT

<EXAMPLE> *(example-caption[\ symbol-name])*

ARGUMENTS

example-caption

Specifies the text of the caption to be associated with the example, if any. The example caption and the associated example number will be written to the table of contents for the document.

symbol-name

Specifies the symbolic identifier to be associated with the example. The symbol identifier will be assigned a numeric value, which will be the current example number. The symbol and its value are placed in the symbol table.

Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores. Do not begin a symbol-name with an underscore.

related tags

- <CODE_EXAMPLE>
- <EXAMPLE_ATTRIBUTES>
- <EXAMPLE_FILE>
- <EXAMPLE_SPACE>
- <INTERACTIVE>
- <VALID_BREAK>

restrictions

Cannot contain <TABLE> , <FIGURE> , or <MATH> tags.

required terminator

<ENDEXAMPLE>

DESCRIPTION

The <EXAMPLE> tag begins an example.

If the body of the example spans more than a single page of text, the example caption is repeated on each page on which the example continues.

If the example is more than a page, page breaks are handled automatically by the text processor. You can control the page breaking by using <PAGE> tags to specify explicit page breaks, or <VALID_BREAK> tags to specify good breaking points.

Note that the presence of the symbol-name argument and the example-caption argument indicate that the example is a formal, numbered example.

When a floating figure or floating example precedes a multipage table (all tables not marked with the KEEP keyword are multipage), a succession of short pages may occur before the page on which the table begins. This is because a multipage table forces any previous floating figures or examples to be output before the table begins.

If this situation occurs, code the preceding floating figures or examples with the KEEP keyword so that they will be kept with the text preceding them, and so result in better-balanced pages.

EXAMPLE

```
<EXAMPLE>(VAXcluster Multi-file Summary\multi_file_exam)
<EXAMPLE_ATTRIBUTES>(WIDE)
<EXAMPLE_FILE>(monitor_multi_file_summary)
<ENDEXAMPLE>
```

This example illustrates a one-page example, which uses the WIDE argument of the <EXAMPLE_ATTRIBUTE> tag. Assume that this example is the first example in the fourth chapter.

The contents of the example are included using the <EXAMPLE_FILE> tag. The file that is included must contain either a code example or interactive example.

<EXAMPLE>

This example may produce the following output:

Example x–x VAXcluster Multi-file Summary

```
            +-----+         VAX/VMS Monitor Utility
            | AVE |         TIME IN PROCESSOR MODES
            +-----+            MULTI-FILE SUMMARY

             Node:      MOE              CURLEY              LARRY
             From: 15-APR-1984 18:17  15-APR-1984 18:17  15-APR-1984 18:17      Row     Row        Row
             To:   15-APR-1984 20:17  15-APR-1984 20:17  15-APR-1984 20:17      Sum  Average   Minimum

Interrupt Stack                 6.51                0.50                6.25     13.2      4.4      0.50
Kernel Mode                    25.73                0.42               12.43     38.5     12.8      0.42
Executive Mode                  9.46                0.95                1.81     12.2      4.0      0.95
Supervisor Mode                 1.97                0.00                0.16      2.1      0.7      0.00
User Mode                      13.24                5.33               56.14     74.7     24.9      5.33
Compatibility Mode              0.00                0.07                0.00      0.0      0.0      0.00
Idle Time                      23.61                0.02               92.63    116.2     38.7      0.02

            +-----+         VAX/VMS Monitor Utility
            | AVE |       PAGE MANAGEMENT STATISTICS
            +-----+            MULTI-FILE SUMMARY

             Node:      MOE              CURLEY              LARRY
             From: 15-APR-1984 18:17  15-APR-1984 18:17  15-APR-1984 18:17      Row     Row        Row
             To:   15-APR-1984 20:17  15-APR-1984 20:17  15-APR-1984 20:17      Sum  Average   Minimum

Page Fault Rate                36.73                8.81                0.49     46.0     15.3      0.49
Page Read Rate                 14.28                4.71                0.00     19.0      6.3      0.00
Page Read I/O Rate              1.20                0.70                0.00      1.9      0.6      0.00
Page Write Rate                 0.00                0.00                0.00      0.0      0.0      0.00
Page Write I/O Rate             0.00                0.00                0.00      0.0      0.0      0.00

Free List Fault Rate            8.60                1.40                0.24     10.2      3.4      0.24
Modified List Fault Rate        5.83                2.29                0.00      8.1      2.7      0.00
Demand Zero Fault Rate         12.96                1.68                0.24     14.8      4.9      0.24
Global Valid Fault Rate         8.10                2.69                0.00     10.8      3.6      0.00
Wrt In Progress Fault Rate      0.00                0.00                0.00      0.0      0.0      0.00
System Fault Rate               4.92                0.53                0.18      5.6      1.8      0.18

Free List Size               7586.30             8630.14             9665.06  25881.5   8627.1   7586.30
Modified List Size             87.69              324.07               32.12    443.8    147.9     32.12
```

<EXAMPLE_ATTRIBUTES>

Specifies attributes for the current example.

FORMAT

<EXAMPLE_ATTRIBUTES> ([*MULTIPAGE* | *KEEP* | *FLOAT*] *[\ WIDE])*

ARGUMENTS

MULTIPAGE

Specifies that the example has multiple elements, and that each element is allowed to break at the start of a new page. This argument is required if the example will be on more than one page. When an example is continued, the example caption is automatically repeated at the beginning of each new page of output.

KEEP

Specifies whether the example is to be kept with the text that immediately precedes it.

FLOAT

Specifies whether the location of a one-page example is allowed to float. FLOAT indicates that if there is not enough room on the current page for the example, the text processor will fill the current page with the text from the source file that follows the <EXAMPLE> tag sequence, and place the example at the top of the next page of output.

Float is the default for an example that has a caption and does not specify MULTIPAGE or KEEP.

WIDE

Specifies that the width of the example exceeds the document's default width for text. Depending on the doctype, this argument can be interpreted as follows:

- If the document style contains a left margin area that is normally used for headings, the example's width will span that area as well as the normal text area.
- If the document uses a multicolumn format, the example suspends multicolumn output while the example is processed. The example is output and the multicolumn output is then restored.
- If the document style provides a range of sizes and styles for examples, this argument can be interpreted to mean that a specific size should be used for the example.

<EXAMPLE_ATTRIBUTES>

related tags

- <EXAMPLE>
- <EXAMPLE_FILE>
- <EXAMPLE_SPACE>

restrictions

Must be enabled by <EXAMPLE> .

required terminator

None.

DESCRIPTION

By default, if an example does not fit on the current output page, it is placed at the top of the next page. The text that follows the example in the SDML file is used to fill the current output page. Thus, the example's position in the text may change. It is said to "float."

If the example will be longer than a page, use the MULTIPAGE argument. The presence of the MULTIPAGE argument indicates that the example requires more than one page of output. The example will not be allowed to float.

Formal examples that are coded with the default FLOAT argument may appear at the bottom of a page preceding text that should normally be at the top of a new page. Use <EXAMPLE_ATTRIBUTES> (KEEP) to force the formal example to be output before the new page.

EXAMPLE

```
<EXAMPLE>(Command Procedure for Adding a User\adduserex)
<EXAMPLE_ATTRIBUTES>(WIDE\MULTIPAGE)
<CODE_EXAMPLE>(WIDE)
$ !
$ !   ADDUSER.COM -- Adds a new user to the system authorization file
$ !
$ USERDISK = "WRKD$:"                ! Default disk for new users
$ UAF = "$AUTHORIZE"
$ ON CONTROLY THEN GOTO CLEANUP
$ ON WARNING THEN GOTO CLEANUP
$ OLDDIR = F$LOGICAL("SYS$DISK") + F$DIRECTORY()'
$ PREVPRIV = F$SETPRV("SYSPRV")
$ IF .NOT. F$PRIVILEGE("SYSPRV") THEN GOTO NOPRIV
$ SET DEFAULT SYS$SYSTEM
$ !
$ ! Request account information
$ !
$ INQUIRE USERNAME "Username"
$ INQUIRE FULLNAME "Full name"
$ SET TERMINAL/NOECHO
$ INQUIRE PASSWORD "Password [''Username']"
$ SET TERMINAL/ECHO
$ IF PASSWORD .EQS. "" THEN PASSWORD = USERNAME
<VALID_BREAK>
$GET_GRP:
$ INQUIRE GRP "UIC Group Number"
$ IF GRP .EQS. "" THEN GRP = "*"
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "Determine the UIC from the following listing:"
$ WRITE SYS$OUTPUT ""
```

```
$ UAF SHOW ['GRP',*]/BRIEF
$ INQUIRE UIC
$ IF UIC .EQS. "" THEN GOTO GET_GRP
$ IF F$LOCATE("[",UIC) .EQ. F$LENGTH(UIC) .AND. -
      F$LOCATE("<",UIC) .EQ. F$LENGTH(UIC) THEN UIC = "[" + UIC + "]"
$ INQUIRE ACCOUNT "Account Name [VMS]"
$ IF ACCOUNT .EQS. "" THEN ACCOUNT = "VMS"
$ INQUIRE PRIVS "Privileges [NONE]"
$ IF PRIVS .NES. "" THEN PRIVS = "/PRIV=(" + PRIVS + ")"
$ USERDIR = F$EXTRACT(0,9,USERNAME)
$ INQUIRE TMP "Login Directory [''USERDIR']"
$ IF TMP .NES. "" THEN USERDIR = TMP
$ INQUIRE TMP "Login Device [''USERDISK']"
$ IF TMP .NES. "" THEN USERDISK = TMP
$ DQUOTA = 0
$ IF F$SEARCH("''USERDISK'[0,0]QUOTA.SYS") .EQS. "" THEN GOTO NQO
$ DQUOTA = 1
<VALID_BREAK>

$ !
$ ! Restore prior working environment
$ !
$CLEANUP:
$ SET TERMINAL/ECHO
$ PREVPRIV = F$SETPRV(PREVPRIV)
$ SET DEFAULT 'OLDDIR'
$ EXIT
$ !
$ ! In case proper privileges are not set
$ !
s$NOPRIV:
$ WRITE SYS$OUTPUT "You need SETPRV or SYSPRV privilege to run this procedure"
$ GOTO CLEANUP
<ENDCODE_EXAMPLE>
<ENDEXAMPLE>
```

This example illustrates the use of the <MULTIPAGE> argument. Two points especially should be noted:

- The <VALID_BREAK> tags in the <CODE_EXAMPLE> provide the text formatter with information about suitable places to break pages.
- Because the example is wide, the WIDE argument was specified not only for the <EXAMPLE> tag, but also for the <CODE_EXAMPLE> tag within the example.

This example may produce the following output:

Example x—x Command Procedure for Adding a User

```
$ !
$ !   ADDUSER.COM -- Adds a new user to the system authorization file
$ !
$ USERDISK = "WRKD$:"                ! Default disk for new users
$ UAF = "$AUTHORIZE"
$ ON CONTROLY THEN GOTO CLEANUP
$ ON WARNING THEN GOTO CLEANUP
$ OLDDIR = F$LOGICAL("SYS$DISK") + F$DIRECTORY()'
$ PREVPRIV = F$SETPRV("SYSPRV")
$ IF .NOT. F$PRIVILEGE("SYSPRV") THEN GOTO NOPRIV
$ SET DEFAULT SYS$SYSTEM
$ !
$ ! Request account information
$ !
$ INQUIRE USERNAME "Username"
$ INQUIRE FULLNAME "Full name"
$ SET TERMINAL/NOECHO
$ INQUIRE PASSWORD "Password [''Username']"
$ SET TERMINAL/ECHO
$ IF PASSWORD .EQS. "" THEN PASSWORD = USERNAME
$GET_GRP:
$ INQUIRE GRP "UIC Group Number"
$ IF GRP .EQS. "" THEN GRP = "*"
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "Determine the UIC from the following listing:"
$ WRITE SYS$OUTPUT ""
$ UAF SHOW ['GRP',*]/BRIEF
$ INQUIRE UIC
$ IF UIC .EQS. "" THEN GOTO GET_GRP
$ IF F$LOCATE("[",UIC) .EQ. F$LENGTH(UIC) .AND. -
      F$LOCATE("<",UIC) .EQ. F$LENGTH(UIC) THEN UIC = "[" + UIC + "]"
$ INQUIRE ACCOUNT "Account Name [VMS]"
$ IF ACCOUNT .EQS. "" THEN ACCOUNT = "VMS"
$ INQUIRE PRIVS "Privileges [NONE]"
$ IF PRIVS .NES. "" THEN PRIVS = "/PRIV=(" + PRIVS + ")"
$ USERDIR = F$EXTRACT(0,9,USERNAME)
$ INQUIRE TMP "Login Directory [''USERDIR']"
$ IF TMP .NES. "" THEN USERDIR = TMP
$ INQUIRE TMP "Login Device [''USERDISK']"
$ IF TMP .NES. "" THEN USERDISK = TMP
$ DQUOTA = 0
$ IF F$SEARCH("''USERDISK'[0,0]QUOTA.SYS") .EQS. "" THEN GOTO NQO
$ DQUOTA = 1
$ !
$ ! Restore prior working environment
$ !
$CLEANUP:
$ SET TERMINAL/ECHO
$ PREVPRIV = F$SETPRV(PREVPRIV)
$ SET DEFAULT 'OLDDIR'
$ EXIT
$ !
$ ! In case proper privileges are not set
$ !
$NOPRIV:
$ WRITE SYS$OUTPUT "You need SETPRV or SYSPRV privilege to run this procedure"
$ GOTO CLEANUP
```

<EXAMPLE_FILE>

Causes a separate file containing an example to be included in the source file.

FORMAT

<EXAMPLE_FILE> *(file-spec)*

ARGUMENTS

file-spec
Specifies the file containing the example.

related tags

- <EXAMPLE>
- <EXAMPLE_SPACE>
- <EXAMPLE_ATTRIBUTES>

restrictions

Cannot be used in an argument to a tag.

required terminator

None.

DESCRIPTION

The <EXAMPLE_FILE> tag causes the entire contents of the file to be included at this point in the SDML file. (It is identical in action to the <INCLUDE> tag.)

The included file should be an SDML file containing a complete coded example, perhaps using <CODE_EXAMPLE> or <INTERACTIVE> tags. If the <EXAMPLE> tag specifies the WIDE argument, the included file must specify the WIDE argument on the appropriate tags. For more information on correct coding of examples and example files, including how to specify valid breaks, see Chapter 3.

The <EXAMPLE_FILE> tag may have a logical name for an argument. The logical name can be defined by first entering an <INCLUDES_FILE> tag in the profile. The information in that tag is used to make the proper logical name assignment during processing.

EXAMPLE

See the example in the discussion of the <EXAMPLE> tag.

<EXAMPLE_SPACE>

Leaves space for an example that will be pasted in during final production.

FORMAT

<EXAMPLE_SPACE> *([value | FULL_PAGE] [\ text])*

ARGUMENTS

value

Specifies the size of the space in picas, a scale used by typesetters. There are approximately 6 picas to the inch. Thus, if the example is 4 inches high, you should specify 24 picas. The value must not exceed page depth limitations for the current document design. If you omit the value, a default value of 2 picas is used.

FULL_PAGE

If you specify the keyword FULL_PAGE, a full blank page is reserved for the example.

text

Specifies text to be printed in the center of the space. (For example, an art file number, a file name that contains the example, or some other note.)

related tags

- <EXAMPLE>
- <EXAMPLE_FILE>
- <EXAMPLE_ATTRIBUTES>
- <REFERENCE>

restrictions

Can be used only in the context of an <EXAMPLE> tag. The value must not exceed page-depth limitations.

required terminator

None.

DESCRIPTION

The <EXAMPLE_SPACE> tag causes vertical space to be left on the output page for an example that will be pasted in by hand during final production. Any text supplied as the second argument is printed in the center of the space.

EXAMPLE

```
<EXAMPLE>(Completing Form DD-214\annotated_dd214_exam)
<EXAMPLE_ATTRIBUTES>(wide)
<EXAMPLE_SPACE>(12\Photo-reduced copy of the end of DD-214 here.)
<ENDEXAMPLE>
```

This example may produce the following output:

Example x—x Completing Form DD-214

Photo-reduced copy of the end of DD-214 here.

<FCMD>

Specifies the keyword portion of a formatted command/parameter pair in a format section.

FORMAT

<FCMD> *(keyword-part[\ parameter-list])*

ARGUMENTS

keyword-part

Specifies the keyword portion of the keyword/parameter pair.

parameter-list

Lists the command parameters, if any.

Parameters specified in this argument will differ on output from arguments specified using <FPARMS> . If you do specify <FCMD> with a null second argument, it is recommended that you explicitly declare the absence of parameters using <FPARMS> as follows:

```
<FCMD>(KEYWORD-PART) <FPARMS>()
```

If you do not specify a second argument, and if <FPARMS> is not specified, SDML will issue a warning message that it has no explicit declaration of parameters.

related tags

- <FORMAT>
- <FPARM>
- <FPARMS>

restrictions

Enabled only within the context of the <FORMAT> tag.

required terminator

None.

DESCRIPTION

The <FCMD> tag identifies a command or keyword within a format section. You can specify the command or keyword's argument list as the second argument to this tag. The <FCMD> tag should be used only in the context of a format section. A format section is established by <FORMAT> and <ENDFORMAT> .

If the parameter-list argument text does not fit on one line, the text formatter chooses suitable line breaks based on the presence of spaces.

EXAMPLES

1
```
<FORMAT>
<FCMD>(exit) <FPARMS>()
<ENDFORMAT>
```

Specifies a single command keyword. <FPARMS> is explicitly specified as null. The output from this code fragment is shown in Output Sample 1.

2
```
<FORMAT>
<FCMD>(append) <FPARMS>(input-file-spec output-file-spec)
<ENDFORMAT
```

This example specifies the command keyword and its parameters using both the <FCMD> and <FPARMS> tags. The output that this example produces is shown in Output Sample 2.

Notice the difference between Output Sample 2 and 3. In Sample 2, space is left between the command-keyword and the parameter-list. In Sample 3, no space is left. This formatting is produced by the syntax used in the <FCMD> command.

3
```
<FORMAT>
<FCMD>(f$element\(element-number,delimiter,string))
<ENDFORMAT
```

This example specifies both the *command-keyword* and the *parameter-list* arguments to the <FCMD> tag. The output that this example produces is shown in Output Sample 3.

4
```
<FORMAT>
<FCMD>(set protection\[=code]<keyword>(/DEFAULT))
<ENDFORMAT
```

This example illustrates how to control the interpretation of keywords tagged with the global <KEYWORD> tag when they are mixed in with the parameter list of a format specification. The output that this example produces is shown in Output Sample 4.

OUTPUT SAMPLE 1

exit

OUTPUT SAMPLE 2

append *input-file-spec output-file-spec*

OUTPUT SAMPLE 3

f$element*(element-number,delimiter,string)*

OUTPUT SAMPLE 4

set protection*[=code]***/DEFAULT**

<FIGURE>

Labels the beginning of a figure.

FORMAT

<FIGURE> *[(figure-caption[\ symbol-name])]*

ARGUMENTS

figure-caption

Specifies the text of the caption to be associated with the figure. The figure-caption and the associated figure number will be written to the table of contents for the document.

symbol-name

Specifies the symbolic identifier to be associated with the example. The symbol-name is assigned a numeric value, which becomes the current figure number. The symbol-name and its value are placed in the symbol table.

All symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

related tags

- <FIGURE_ATTRIBUTES>
- <FIGURE_FILE>
- <FIGURE_SPACE>
- <LINE_ART>
- <REFERENCE>
- <ICON>
- <ICON_FILE>

restrictions

None.

required terminator

<ENDFIGURE>

DESCRIPTION

The <FIGURE> tag is used to label formal and informal figures. A formal figure is listed in the table of contents, has a unique number by which it is cross-referenced from other parts of your book, and has a caption that labels it. An informal figure does not appear in the table of contents and has no caption or number identifier.

The presence of a figure-caption and symbol-name indicate that the figure is a formal, numbered figure.

The figure-caption argument of this tag specifies the text of the caption associated with the figure. The figure caption and the associated figure number will be written to the table of contents for the document. If the body of the figure spans more than a single page of text, the figure caption is repeated on each page on which the figure continues.

The symbol-name specifies the symbolic identifier to be associated with the figure. The symbol identifier will be assigned a numeric value, which will be the current figure number. The symbol and its value are placed in the symbol table.

A formal figure is sometimes long. If you think one of your figures will be longer than a page, label it as a multipage figure in the <FIGURE_ATTRIBUTES> tag.

If the figure does not fit on the current output page, the figure is placed at the top of the next page. The text that follows the example in the SDML file is used to fill the current output page. Thus, the figure's position in the text may change or "float." If floating is undesirable you must use the KEEP argument to fix the figure at its current position in the text. If it does not fit on the page, the page is terminated, and the figure begins on a new page. Use of KEEP ensures that the figure and its accompanying text remain in the same order. However, KEEP should be used sparingly and only where examples are short and frequent, in order to avoid excessively short pages.

When a floating figure or floating example precedes a multipage table (all tables not marked with the KEEP keyword are multipage), a succession of short pages may occur before the page on which the table begins. This is because a multipage table forces any previous floating figures or examples to be output before the table begins.

If this situation occurs, code the preceding floating figures or examples with the KEEP keyword so that they will be kept with the text preceding them, and so result in better-balanced pages.

EXAMPLES

1

```
<FIGURE>(Example of Terminal-Created Art\transfer_vector_exam)
<FIGURE_ATTRIBUTES>(KEEP)
<LINE_ART>(KEEP)

               Shareable Image           Executable Image
                +-----------+             +------------+
       +----| Vector A  |<-------+     |      .     |
            |   +-----------+         |    |      .      |
            | +--| Vector B  |<---+    +----- Invoke A    |
            | |  +-----------+     |        |      .      |
           +--->|           |     |        |      .      |
              |  | Routine A |    +--------- Invoke B    |
              |  |           |               |      .      |
              |  +-----------+               |      .      |
             +->|           |               +------------+
                 | Routine B |
                 |           |
                +-----------+

<ENDLINE_ART>
<ENDFIGURE>
```

In this example, the <FIGURE_ATTRIBUTES> tag contains a KEEP argument, to specify that the figure should be kept with the immediately preceding text.

The <LINE_ART> tag also has a KEEP argument, to specify that the figure should not break across pages.

The figures you are able to create using your terminal keyboard may be adequate for draft purposes, but will not be acceptable in the context of typeset output. This example may produce the following output:

Figure x—x Example of Terminal-Created Art

```
   Shareable Image           Executable Image
     +-----------+             +------------+
+----| Vector A  |<-------+    |      .     |
|    +-----------+        |    |      .     |
| +--| Vector B  |<---+   +----- Invoke A   |
| |  +-----------+    |        |      .     |
+--->|           |    |        |      .     |
  |  | Routine A |    +-------- Invoke B    |
  |  |           |             |      .     |
  |  +-----------+             |      .     |
  +->|           |             +------------+
     | Routine B |
     |           |
     +-----------+
```

2

```
<DEFINE_SYMBOL>(ZK_2347\2)
<FIGURE>(Illustration of the Workshop Layout Showing Forklift\ZK_2347)
<FIGURE_SPACE>(FULL_PAGE\An illustration of the memory management workshop
analogy, complete with picture of the forklift.)
<ENDFIGURE>
```

This example illustrates how to code a figure that will take one page. This example may produce output like that found on the following page.

Figure x—x Illustration of the Workshop Layout Showing Forklift

An illustration of the memory management workshop analogy complete with picture of the forklift.

3

```
<FIGURE>(Discreet Portions of the Process Header\proc_head_fig)
<FIGURE_ATTRIBUTES>(Multipage)
<FIGURE_SPACE>(15\Shows general structure of process header)
<FIGURE_SPACE>(15\Shows structure of process page tables)
<FIGURE_SPACE>(15\Shows structure of different forms of page table entry)
<ENDFIGURE>
```

This example shows how to code a figure that is allowed to extend over several pages. The MULTIPAGE argument to the <FIGURE_ATTRIBUTES> tag tells the text formatter that the figure is more than one page.

<FIGURE>

Figure x—x Discrete Portions of the Process Header

Shows general structure of process header

Figure x—x Cont'd. on next page

<FIGURE>

Figure x—x (Cont.) Discrete Portions of the Process Header

Shows structure of process page tables

Figure x—x Cont'd. on next page

<FIGURE>

Figure x—x (Cont.) Discrete Portions of the Process Header

Shows structure of different forms of page table entry

4
```
<P> ... and then procedes to linking the object
modules into an executable image. (You are not required to
put object modules in a library. You can link them directly.) The following
figure illustrates the implementation cycle.

<FIGURE>
<FIGURE_SPACE>(17\ZK--8769--84)
<ENDFIGURE>
<P>
The first cycle shows that  . . .
```

This example shows how to code an informal figure that appears within text without floating, without a number, and without a caption. It may produce the following output:

... and then proceeds to linking the object modules into an executable image. (You are not required to put object modules in a library. You can link them directly.) The following figure illustrates the implementation cycle.

ZK–8769–84

The first cycle shows that . . .

<FIGURE_ATTRIBUTES>

Specifies the placement of a figure on the page. Also specifies valid page breaks for that figure.

FORMAT

<FIGURE_ATTRIBUTES> ([*MULTIPAGE* | *KEEP* | *FLOAT*] [\ *WIDE*])

ARGUMENTS

MULTIPAGE

Specifies that the figure has multiple elements, and that each element is allowed to break at the start of a new page. This argument is required if the figure will be on more than one page. When a figure is continued, the figure caption and figure column heads, if any, are automatically repeated at the beginning of each new page of output.

KEEP

Specifies that the figure is to be kept with the immediately preceding text. This is the default for a figure without a caption.

FLOAT

Specifies whether the location of a one-page figure is allowed to float. FLOAT indicates that if there is not enough room on the current page for the figure, the text processor will fill the current page with the text from the source file that follows the <FIGURE> tag sequence, and place the figure at the top of the next page of output.

Float is the default for a figure that has a caption and does not specify MULTIPAGE or KEEP.

WIDE

Specifies that the width of the figure exceeds the document's default width for text. Depending on the document type, this argument can be interpreted as follows:

- If the document style contains a left margin area that is normally used for headings, the figure's width will span that area as well as the normal text area.
- If the document uses a multicolumn format, the figure suspends multicolumn output while the figure is processed. The figure is output and multicolumn output is then restored.
- If the document style provides a range of sizes and styles for figures, this argument may be interpreted to mean that a specific size should be used for the figure.

<FIGURE_ATTRIBUTES>

related tags

- <FIGURE>
- <FIGURE_FILE>
- <FIGURE_SPACE>

restrictions

This tag must be used within the context of a <FIGURE> tag.

required terminator

None.

DESCRIPTION

This tag is used in the context of a formal figure. Use <FIGURE_ATTRIBUTES> to adjust the pagination and placement of the figure.

A formal figure is sometimes long. If you think one of your figures will be longer than a page, use the MULTIPAGE argument. Your figure will then be placed on the current page and continued on following pages, as space permits.

A multipage figure can contain one or more elements (that is, one or more <FIGURE_FILE> , <FIGURE_SPACE> , and <CODE_EXAMPLE> tags).

If your multipage figure (consisting of a single <CODE_EXAMPLE>) contains no blank lines, you might want to insert <VALID_BREAK> tags in the figure. The <VALID_BREAK> tags specify reasonable page breaking points. See the description of the <VALID_BREAK> tag for more information.

A one-page (or smaller) figure's position in the output file can change. If the figure does not completely fit on the current page, the entire figure is placed at the top of the next page. The text that follows the example in the SDML file then is used to fill the current output page. When the figure's position can change, the figure is said to *float*.

Floating is the default condition for a figure that has a caption and fits on one page. If you want the placement of a figure without a caption to float, you must specify the FLOAT argument if your figure fits on one page. A multipage figure never floats.

The argument KEEP will override the default FLOAT condition for a figure with a caption.

Figures that are coded with the default FLOAT argument may appear at the bottom of a page preceding text that should normally be at the top of a new page. Use <FIGURE_ATTRIBUTES> (KEEP) to force the figure to be output before the new page.

EXAMPLE

See the first example in the <FIGURE> tag.

<FIGURE_FILE>

Includes a graphics file in your output file if the output device has graphics capability.

FORMAT

<FIGURE_FILE> *(target-device*
{ *\ file-spec* }
{ *\ SPACE* }
\ vertical-size
[\ position])

ARGUMENTS

target-device

Specifies a keyword indicating the output device for the graphics file. Keywords are provided both for devices that do (LN03 and PS) and do not (LINE, MAIL, and TERMINAL) support graphics. Each keyword allows you to insert the necessary amount of white space for the specific output device; keywords for devices that do not support graphics allow you to insert blank space in place of a figure. The following table lists the output devices and expected output for each keyword.

Keyword	Device	Output
LN03	LN03 Laser Printer	The specified sixel graphics file is output as a figure.
PS	PRINTSERVER 40 or LN03R SCRIPTPRINTER	The specified POSTSCRIPT graphics file[1] is output as a figure.
LINE MAIL TERMINAL	Line Printer	If specified, blank space is output with the file-spec argument written in that blank space. Only one of these three keywords should be used to indicate a monospaced destination.

[1] POSTSCRIPT graphics files must conform to the Encapsulated POSTSCRIPT File Format published by Adobe Systems Incorporated.

If you specify <FIGURE_FILE> for a given device and subsequently process the file on another output device, no output will appear in the position of the <FIGURE_FILE> tag.

file-spec

Specifies the graphics file.

SPACE

Lets you reserve blank space in the output file for a figure. Use this keyword to reserve space for art that will be pasted in at a later date or when you expect to process a file for more than one output device, but do not have graphics files for all devices.

vertical-size

Specifies the vertical size of the printed graphic in picas (six picas equal an inch). This argument may be a nonnegative integer or decimal number, including zero.

position

Specifies an optional keyword that determines how the graphic aligns in relationship to the text:

- null — aligns with the normal left text margin.
- WIDE — aligns at the leftmost position of the image area (if your format has a wide left gutter).

related tags

- <ICON>
- <ICON_FILE>
- <FIGURE>
- <SET_FIGURE_FILE_SPACING_DEFAULT>

restrictions

You must use an LN03 or POSTSCRIPT laser printer to print graphics files.

You must use the <FIGURE_FILE> tag in the context of the <FIGURE> tags.

If you use the <FIGURE_FILE> tag in a file that you process for multiple output devices, you must supply a tag for each device, using the SPACE keyword for devices for which you do not have graphics.

required terminator

None.

DESCRIPTION

The <FIGURE_FILE> tag enables you to automatically include graphics files for printing along with your text in the output file.

To ensure that your figure is the right size for inclusion with your output, follow this procedure:

1. Create and print your graphics file full size (100 percent).
2. Measure it.
3. If the figure at 100 percent is the desired size and will fit on the page, incorporate that measurement into the <FIGURE_FILE> tag. Otherwise, calculate a percentage reduction and use the graphics editor to reduce the figure to the desired size. Print and measure the graphics file and incorporate the measurement in the <FIGURE_FILE> tag.
4. Process the SDML file and print the output.

5 Evaluate the result. If it is satisfactory, you are finished. Otherwise, change either of the following.

- One or both keywords specified in <FIGURE_FILE> to correct the alignment, spacing, or both, for the figure.
- The graphics file itself to correct the size, content, or both

Because there is an interaction between changing the contents or size of the graphics file and specifying its placement in the <FIGURE_FILE> tag, satisfactory results may require several loops through the process, much like coding complicated tables. Refer to Chapter 3 for information on coding tables.

If you plan to process your file for more than a single destination, you can use multiple <FIGURE_FILE> tags to ensure that the appropriate figure will be included for the appropriate destination; an example of this coding is given in the following "Examples" section. If you use multiple <FIGURE_FILE> tags in this way, you should specify only a single <FIGURE_FILE> tag for all the monospaced destination keywords (MAIL, TERMINAL, or LINE).

EXAMPLES

1

```
<FIGURE>(The Front View)
<FIGURE_FILE>(LN03\frontpanel.six\6)
<ENDFIGURE>
```

This example shows how to code a numbered figure.

2

```
<FIGURE>
<FIGURE_FILE>(LN03\example1.six\10)
<ENDFIGURE>
```

This example shows how to code an unnumbered figure.

3

```
<FIGURE>
<FIGURE_FILE>(LN03\SANTA.SIX\15)
<FIGURE_FILE>(PS\SANTA.PS\15)
<FIGURE_FILE>(LINE\SPACE\10)
<ENDFIGURE>
```

This example shows how to use multiple <FIGURE_FILE> tags so that your figure will process for multiple destinations. If this file is processed for a POSTSCRIPT destination, the file included would be SANTA.PS. If this file is processed for an LN03 laser printer destination, the file included would be SANTA.SIX. If this file is processed for a line printer destination no file would be included, but instead 10 blank lines would be reserved for the graphic.

A sixel file included into a document as in the previous code example may have the following output:



ZK-7713-HC

<FIGURE_SPACE>

Marks the space required for a figure that will be pasted in during final production.

FORMAT

<FIGURE_SPACE> ({ *value* | *FULL_PAGE* } [\ *text*])

ARGUMENTS

value

Specifies the amount of vertical space to be left on the page. The value should be specified in picas, a scale used by typesetters. There are approximately 6 picas to the inch. Thus, if the figure to be pasted in is 4 inches high, you should specify 24. If you do not specify a value, a default value of 2 is used.

FULL_PAGE

Specifies that a full blank page is reserved for the figure.

text

Specifies text that describes the status of the figure, an art file number or the words "To Be Set." The text is output in the middle of the space left for the figure.

related tags

- <FIGURE>
- <FIGURE_ATTRIBUTES>
- <FIGURE_FILE>
- <REFERENCE>
- <FIGURE_SPACE>
- <LINE_ART>

restrictions

Must be used in the context of a <FIGURE> tag.

The value of the <FIGURE_SPACE> tag must not exceed page depth limitations.

required terminator

None.

<FIGURE_SPACE>

DESCRIPTION

The <FIGURE_SPACE> tag causes a blank space to be left on the page for a figure that will be pasted in by hand during final production.

If you specify some descriptive text in the second argument, that text is output in the middle of the space left for the figure.

EXAMPLE

See the example in the discussion of the <FIGURE> tag.

<FILE_SPEC>

Allows you to use a file specification that contains angle brackets as an argument to an SDML tag without VAX DOCUMENT interpreting that file specification as an SDML tag.

FORMAT

<FILE_SPEC> *(file-specification)*

ARGUMENTS

file-specification

The file specification that contains angle brackets.

related tags

- <ELEMENT>
- <EXAMPLE_FILE>
- <FIGURE_FILE>
- <ICON_FILE>
- <INCLUDE>
- <INCLUDES_FILE>
- <TABLE_FILE>

restrictions

None.

DESCRIPTION

The <FILE_SPEC> tag allows you to use a file specification containing angle brackets as the file-spec argument to SDML tags that let you include external files, such as <INCLUDE> and <FIGURE_FILE>. A complete list of such tags is found in the "Related Tags" section of this tag description.

Some keyboards do not provide square brackets ([) and (]), so it is common for the directory portion of a file specification to be delimited by angle brackets. Because VAX DOCUMENT would normally interpret text within angle brackets as an SDML tag, the <FILE_SPEC> tag converts any angle brackets in its argument into the corresponding square brackets.

You do not have to use the <FILE_SPEC> tag to specify logical names in a file specification whose equivalence strings contain angle brackets.

EXAMPLE

```
<figure_file>(ln03\<file_spec>(mydisk:<figure>diagram.six)\20)
```

The directory name in this file specification would normally be seen as a <FIGURE> tag, which would lead to erroneous results. Because the file specification is used as an argument to the <FILE_SPEC> tag, the angle brackets will be converted to square brackets before the apparent <FIGURE> tag is recognized. Thus, the example is equivalent to the following.

```
<figure_file>(ln03\mydisk:[figure]diagram.six\20)
```

<FINAL_CLEANUP>

Provides explicit formatting instructions for the text formatter to be used for final formatting and cleanup.

FORMAT

<FINAL_CLEANUP> ({ *COLUMN_BREAK* | *LINE_BREAK* | *PAGE_BREAK* | *SPECIAL_BREAK* })

ARGUMENTS

COLUMN_BREAK

Specifies that a new column of text is to be started at the place where the <FINAL_CLEANUP> tag occurs. This argument is only valid in a two-column design (ARTICLE or REPORT.TWOCOL) and should be used only to make the final documentation look better.

LINE_BREAK

Specifies that the text formatter is to place the remaining text in the paragraph onto a new line of output.

PAGE_BREAK

Specifies that the text formatter should place following text on a new page of output.

SPECIAL_BREAK

Specifies a special break when vertical spacing appears to be lost. In some circumstances, the output of a two-column page may have had some of its vertical spacing lost due to the text formatter processing, for example, a heading tag may not have any space before it. The SPECIAL_BREAK argument should be used only if there is a spacing problem and only after you are ready to give your document a final format check, because changes to the source file may help the text formatter resolve the spacing problem.

related tags

- <COLUMN> — ARTICLE and REPORT.TWOCOL
- <LINE>
- <PAGE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <FINAL_CLEANUP> tag is not a generic markup tag; it explicitly instructs the text formatter to change aspects of a page makeup.

In a single-column doctype, the <FINAL_CLEANUP> tag enables you to specify line breaks and page breaks using the LINE_BREAK and PAGE_BREAK keyword arguments. In a two-column doctype, you can use the COLUMN_BREAK keyword argument to start a new column of text. The SPECIAL_BREAK keyword argument is used only rarely at the final stage of production to place "finishing touches" into your document.

When you are working with the ARTICLE or REPORT.TWOCOL doctypes, you may need to make some final adjustments when your text is complete. The text formatter makes formatting decisions based on your source file. It is more difficult for the text formatter to create a well-formatted two-column page than it is a well-formatted one-column page. Therefore, the output of these doctypes may need additional "finishing touches" during the last stage of document production.

<FOOTNOTE>

Places a footnote character in text, using the character specified in the tag's argument, and places the footnote text at the bottom of the page.

FORMAT

<FOOTNOTE> *(char \ footnote-text)*

ARGUMENTS

char

Specifies the footnote character. The character can be a single character, a number, or one of the following keywords denoting special characters associated with footnote references:

DAG (†)
DDAG (‡)
R (®)
S (§)
TM (™)

When placing a footnote in a table, you must use a number or one of the keywords as the footnote character.

If you use more than one of the keywords allowed for table footnotes, you should declare them in the following order to ensure that the footnotes print in the correct order at the bottom of the page:

1. TM
2. R
3. S
4. DAG
5. DDAG

footnote-text

Specifies the text of the footnote.

related tags

- <FOOTREF>
- <FOOTNOTE_TEXT>

restrictions

Invalid in math.

No more than four footnotes may be placed within a stacked list or a monospaced example.

The following restrictions apply to footnotes within tables:

- There can be no more than twelve footnotes in a table. If you use a nonnumeric footnote character, there can be no more than seven numeric footnotes.

- All table footnotes must be declared at the top of the table using a <FOOTNOTE> tag just after the <TABLE_SETUP> tag. <FOOTNOTE> tags must not be specified in a nested table.
- Callouts within the body of a table must be labeled with a <FOOTREF> tag. It is the <FOOTREF> tag that causes the footnote character to appear.
- Footnotes in the body of a multipage table appear at the bottom of a page only if they are called out on that page.
- Footnotes called out in the heading of a multipage table appear at the bottom of each page of the table.
- Footnotes on a title page and copyright page should be specified using the <FOOTNOTE_TEXT> tag.

required terminator

None.

DESCRIPTION

The <FOOTNOTE> tag causes a footnote character to appear in text at the place where the tag is located. The text of the footnote is specified as the second argument to the tag, and appears at the bottom of the page.

EXAMPLES

1

```
<P>The <TAG>(footnote) tag may produce output that looks like
this.<FOOTNOTE>(1\Note how
footnote text appears at the bottom of the page.)
```

This example may produce the following output:

The <FOOTNOTE> tag may produce output that looks like this.[1]

2

```
<TABLE>(Rules for Determining Expression Modes\express_modes_tab)
  <TABLE_ATTRIBUTES>(MULTIPAGE)
  <TABLE_SETUP>(2\43)
  <FOOTNOTE>(1\A footnote in a table.)
  <TABLE_HEADS>(Expression\Value Type)
  <TABLE_ROW>(Integer value\Integer)
  <TABLE_ROW>(String value\String<FOOTREF>(1))
  <TABLE_ROW>(Integer lexical function\Integer)
  <TABLE_ROW>(String lexical function\String)
  <TABLE_ROW>(Integer symbol\Integer)
  <TABLE_ROW>(String symbol\String)
  <TABLE_ROW>(Any value .AND. or .OR. any value\Integer)
  <TABLE_ROW>(Any value\Integer)
  <TABLE_ROW>(Any value\Integer)
  <ENDTABLE>
```

This example shows how to produce a two-column table that contains a footnote. A <FOOTNOTE> tag is placed directly after the <TABLE_SETUP> tag to declare the existence of a footnote and to identify the footnote text. A <FOOTREF> tag then is placed in the exact location of the footnote superscripted reference mark in the table.

Make sure to place the <FOOTREF> tag within the argument to a <TABLE_ROW> tag.

[1] Note how footnote text appears at the bottom of the page.

The example may produce the following output:

Table x—x Rules for Determining Expression Modes

Expression	Value Type
Integer value	Integer
String value	String[1]
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
Any value .AND. or .OR. any value	Integer
Any value	Integer
Any value	Integer

[1]A footnote in a table.

<FOOTNOTE_TEXT>

Specifies the text of a footnote in the context of a heading, title page, or copyright page.

FORMAT

<FOOTNOTE_TEXT> *(char \ text)*

ARGUMENTS

char

Specifies the footnote character. The character can be a single character or number, or it can be one of the following keywords denoting special characters associated with footnote references:

DAG (†)
DDAG (‡)
R (®)
S (§)
TM (™)

text

Specifies the text associated with the footnote.

related tags

- <FOOTREF>
- <FOOTNOTE>

restrictions

Required when footnotes are specified in the context of header-level text and title pages/copyright pages. Must not be used for table footnotes. The <FOOTNOTE_TEXT> tag must follow the related <FOOTREF> tag in the source file.

required terminator

None.

DESCRIPTION

The tag <FOOTNOTE_TEXT> allows you to specify the text, placement, or both, of a footnote in the following special circumstances:

- When you want to specify a footnote in an argument to a header-level tag (<HEAD1> , <HEAD2> , etc.)
- When you want to specify footnote text related to a footnote character on the title page of a document.

In all other cases, you should use the <FOOTNOTE> tag.

EXAMPLES

1

```
<HEAD1>(Introduction to The News Today<footref>(TM))
<footnote_text>(TM\The News Today
is a trademark of the American Television
Society.)
```

This example illustrates the use and placement of the <FOOTNOTE_TEXT> tag and its related <FOOTREF> .

2

```
<front_matter>
<TITLE_PAGE>
<product>(CATCHUP<footref>(TM))
<title>(Guide to Growing Premium Fruit Bearing Plants)
<ABSTRACT>(June 1986)
This document describes how to cultivate smooth-skinned
tomato plants.
<ENDABSTRACT>

<footnote_text>(TM\CATCHUP is a trademark of Tomato Magnates, Inc.)
<ENDTITLE_PAGE>
```

This example illustrates the placement of a footnote on the title page of a document. The exact placement of the footnote, in the final output, is based on the document-specific design specified by the book designer.

<FOOTREF>

Creates one or more footnote characters in text or in a table using the footnote numbers or characters as arguments.

FORMAT

<FOOTREF> *(char-1 [\ char-2... \ char-9])*

ARGUMENTS

char-1...9

The character(s) for the footnote. A character can be a single character or number, or it can be one of the following special characters associated with footnote references:

DAG (†)
DDAG (‡)
R (®)
S (§)
TM (™)

In a table, the reference must be to a number or to one of the special characters.

related tags

- <FOOTNOTE>
- <FOOTNOTE_TEXT>

restrictions

Invalid in math.

Can only be used at the end of an argument list and cannot be embedded in text; the end of an argument list is indicated by a backslash (\) or a closing parenthesis ( )).

required terminator

None.

DESCRIPTION

The <FOOTREF> tag causes as many as nine superscripted characters to appear in the text at the place where the tag is located in the SDML file.

There are two occasions that require use of the <FOOTREF> tag:

- Where the same footnote in text needs to be called out more than once
- Wherever a footnote is called out in a table

If you have a footnote in the text to which you want to refer more than once, you should use a <FOOTNOTE> tag to label the first occurrence of the footnote and the <FOOTREF> tag to label the subsequent characters to that note.

The text formatter does not automatically repeat text footnotes if references are output on more than one page.

If you have one or more footnotes within a table, you must declare the footnotes using the <FOOTNOTE> tag just after the <TABLE_SETUP> tag. You can then use the <FOOTREF> tag to label the characters within the table body. Within a table, the presence of the <FOOTREF> tag causes the text of the footnote to appear at the bottom of the table, rather than at the bottom of the page. If the table is longer than a single page, and if the <FOOTREF> tag appears multiple times, the footnote text appears at the bottom of the table on each page containing the reference.

Make sure to place the <FOOTREF> tag within the argument to a <TABLE_ROW> tag.

EXAMPLE

```
<P>The macro format for a $GETDVI request is:<FOOTNOTE>(1\The
eighth (last)
argument is not used; it is reserved for future use.)
<CODE_EXAMPLE>
 $GETDVI [efn],[chan],[devnam],itmlst,[iosb],[astadr],[astprm]
<ENDCODE_EXAMPLE>
<P>The high-level language format for a $GETDVI request is:<FOOTREF>(1)
<CODE_EXAMPLE>
 SYS$GETDVI([efn],[chan],[devnam],itmlst,[iosb],[astadr],[astprm])
<ENDCODE_EXAMPLE>
```

This example may produce the following output:

The macro format for a $GETDVI request is:[1]

```
$GETDVI [efn],[chan],[devnam],itmlst,[iosb],[astadr],[astprm]
```

The high-level language format for a $GETDVI request is:[1]

```
SYS$GETDVI([efn],[chan],[devnam],itmlst,[iosb],[astadr],[astprm])
```

[1] The eighth (last) argument is not used; it is reserved for future use.

<FORMAT>

Enables <FCMD> , <FPARMS> , and <FPARM> to distinguish formatted command keywords and parameters.

FORMAT

<FORMAT> *[([Heading-text | WIDE])]*

ARGUMENTS

heading-text

Specifies a heading.

WIDE

Indicates, for document styles in which the left margin is indented, that the body of the formatted text should be extended into the margin.

related tags

- <FCMD>
- <FPARM>
- <FPARMS>

restrictions

Invalid in math, monospaced examples, tables, and figures.

required terminator

<ENDFORMAT>

EXAMPLE

See the examples in the discussion of the <FCMD> tag.

<FPARM>

Specifies a parameter to be formatted following <FPARMS> , aligned under the parameter list portion of a keyword/parameter list pair.

FORMAT

<FPARM> *(parameter-list)*

ARGUMENTS

parameter-list
Lists additional command parameters, if any.

related tags

- <FCMD>
- <FORMAT>
- <FPARMS>

restrictions

Enabled only within <FORMAT> .

required terminator

None.

EXAMPLE

See the examples in the discussion of the <FCMD> tag.

<FPARMS>

Specifies the parameter portion of a formatted command/parameter pair in a format section.

FORMAT

<FPARMS> *(parameter-list)*

ARGUMENTS

parameter-list
Lists the command parameters, if any. If there are no parameters, you can specify the argument as null: <FPARMS> .

related tags

- <FCMD>
- <FORMAT>
- <FPARM>

restrictions

Enabled only within <FORMAT> .

required terminator

None.

EXAMPLE

See the examples in the discussion of the <FCMD> tag.

<FRONT_MATTER>

Begins the front matter of a book.

FORMAT

<FRONT_MATTER> *[(symbol-name)]*

ARGUMENTS

symbol-name

Specifies the term that you assign to the front matter. A symbol-name argument is required if the front matter is to be part of a bookbuild.

Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

related tags

- See the list of tags in the description.

restrictions

None.

required terminator

<ENDFRONT_MATTER>

DESCRIPTION

The <FRONT_MATTER> tag enables the tags that create the front matter of a book. The following tags are used to create the front matter:

- <ABSTRACT>
- <CONTENTS_FILE>
- <COPYRIGHT_DATE>
- <COPYRIGHT_PAGE>
- <FRONT_MATTER>
- <ORDER_NUMBER>
- <PREFACE>
- <PREFACE_SECTION>
- <PRINT_DATE>
- <REVISION_INFO>
- <TITLE>
- <TITLE_PAGE>

Each of these tags is listed alphabetically within this chapter. The order that these tags are used is shown in the following example.

EXAMPLE

```
<FRONT_MATTER>(front)
<TITLE_PAGE>
<TITLE>(My Latest Book)
<ORDER_NUMBER>(xx-12345)
<ABSTRACT>
This book describes the latest changes to any product.
<ENDABSTRACT>
<REVISION_INFO>(Revision/Update Information:\This is a new manual.)
<ENDTITLE_PAGE>
<COPYRIGHT_PAGE>
<PRINT_DATE>(March 1987)
<COPYRIGHT_DATE>(1987)
<ENDCOPYRIGHT_PAGE>
<PREFACE>(11)
<PREFACE_SECTION>(The changes to your system)
<ENDPREFACE>
<ENDFRONT_MATTER>
```

This example shows the order of all the front matter tags, and how they could be used in a file that contains the front matter of a book.

<GDEF>

Begins the text that defines a term in a glossary.

FORMAT

<GDEF> *(definition)*

ARGUMENTS

definition

Specifies the definition of the term.

related tags

- <GLOSSARY>
- <GTERM>
- <GREF>

restrictions

Can only be used between <GLOSSARY> and <ENDGLOSSARY> tags.

required terminator

None.

DESCRIPTION

See the description of the <GLOSSARY> tag.

EXAMPLE

See the example in the discussion of the <GLOSSARY> tag.

<GLOSSARY>

Formats a glossary of terms in a document or book.

FORMAT

<GLOSSARY> *(text[\ symbol-name])*

ARGUMENTS

text

Specifies any text string that you want to label the start of the glossary. If no text is specified, the default text is the term "Glossary."

symbol-name

Specifies the term that you assign to the glossary and then use to reference it.

All symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

related tags

- <GTERM>
- <GDEF>

restrictions

The symbol-name argument is required if this file will be included as part of a bookbuild.

required terminator

<ENDGLOSSARY>

DESCRIPTION

To create a glossary of terms, use the <GLOSSARY> tag to establish the beginning format. Follow this with a list of <GTERM> tags, with the glossary words defined as arguments to the <GTERM> tags.

Typically, a definition of each glossary term follows the term itself. Create each definition with a <GDEF> (definition) tag.

Remember to terminate the glossary with an <ENDGLOSSARY> tag.

EXAMPLE

```
<Glossary>
<gterm>(habitat)
<gdef>(An area or natural environment.)
<gterm>(habitual)
<gdef>(Acting according to habit.)
<gterm>(hack)
<gdef>(A worn-out horse.)
<gterm>(hackneyed)
<gdef>(Trite, banal.)
<endglossary>
```

This example produces the following output:

Glossary

habitat: An area or natural environment.

habitual: Acting according to habit.

hack: A worn-out horse.

hackneyed: Trite, banal.

<GREF>

Marks a cross-reference to a term within a glossary.

FORMAT

<GREF> *(glossary-term)*

ARGUMENTS

glossary-term
Specifies the term referred to.

related tags

- <GLOSSARY>
- <GTERM>
- <GDEF>
- The following tags label other types of cross-references:

  <REFERENCE>
  <CALLOUT_REF>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <GREF> tag marks a cross-reference to a term within a glossary. The glossary-term argument appears in an italic typeface in some document designs.

<GTERM>

Labels a term to be defined in a glossary.

FORMAT

<GTERM> *(term)*

ARGUMENTS

term
Specifies the term to be defined in the glossary.

related tags

- <GLOSSARY>
- <GDEF>
- <GREF>

restrictions

Can only be used between <GLOSSARY> and <ENDGLOSSARY> tags.

required terminator

<GDEF>

DESCRIPTION

The <GTERM> tag labels a term to be defined in a glossary.

EXAMPLE

See the example in the tag <GLOSSARY> .

<HEADx>

Marks a heading of the level specified (1 through 6).

FORMAT

<HEADx> *(heading-text [\ symbol-name])*

ARGUMENTS

heading-text

Specifies the text of the heading. If the book design you are using produces headings that are all capital letters in your output, those letters will appear that way regardless of how you enter them in your input file. You should, however, use uppercase and lowercases letters according to your local conventions in order to obtain the proper capitalization of the heading in the table of contents and in cross-references.

symbol-name

The name of the symbol to be used in all references to this heading and the text following. Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

related tags

- <SUBHEAD1>
- <SUBHEAD2>
- <CHEAD>

restrictions

None.

required terminator

None.

DESCRIPTION

Each of the six tags, <HEAD1> , <HEAD2> , <HEAD3> , and <HEAD4> , <HEAD5> , and <HEAD6> does the following:

- Outputs the heading text passed to it in its first argument
- Automatically numbers the heading
- Resets all of the counters for lower heading levels (if any)
- Specifies the symbol-name with which cross-references to that heading should be made

Entries for each of the headings may appear in the table of contents, depending on the doctype design.

The proper choice of the heading level depends on an understanding of the logical structure of the document you are writing. A <HEAD2> tag will always be logically subordinate to a <HEAD1> tag. The same is true for the relationship between <HEAD3> and <HEAD2>; <HEAD4> and <HEAD3>; and so on.

EXAMPLES

1

```
<HEAD1>(Running Tasks\tasks)
```

This tag labels a first-level heading.

2

```
<HEAD2>(SET and SHOW Commands\set_show_sec)
```

This tag labels a second-level heading.

You could make a cross reference to this heading by using the tag <REFERENCE> (set_show_sec). In this example, if the heading was the second sublevel of the first section in the document, the reference would be output as Section 1.2.

To control the output of the reference, use one of the formats listed in the following table:

Reference	**Output**
<REFERENCE> (set_show_sec)	Section 1.2
<REFERENCE> (set_show_sec\value)	1.2
<REFERENCE> (set_show_sec\text)	Set and Show Commands
<REFERENCE> (set_show_sec\full)	Section 1.2, Set and Show Commands

<HELLIPSIS>

Places horizontal ellipsis points on a line.

FORMAT

<HELLIPSIS>

related tags

- <ELLIPSIS>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <HELLIPSIS> tag places horizontal ellipsis points on a line. Often, it is used to label omitted material.

EXAMPLE

```
<P>A horizontal ellipsis may provide an indefinite ending <HELLIPSIS>
```

This example may produce the following output:

A horizontal ellipsis may provide an indefinite ending . . .

<HYPHENATE>

Provides information about legal hyphenation of a word of text.

FORMAT

<HYPHENATE> *(part1 \ part2[\ ...part9])*

ARGUMENTS

part1...part9

Specifies the word of text and its valid hyphenation points. Each argument to the tag specifies a portion of the word. Use the argument delimiter, the backslash, to indicate the hyphenation points.

related tags

- <FINAL_CLEANUP> —LINE_BREAK
- <LINE>
- <KEEP>

restrictions

This tag is invalid in math and in monospaced examples.

DESCRIPTION

Use this tag when you are not satisfied with line breaks within paragraphs in your final output and you determine that the line breaks are caused because a word is not being hyphenated. This tag is also useful when a word or term (usually a technical term that is not commonly used) is not being hyphenated correctly.

This tag does not force a term to be hyphenated; it merely provides the text formatter with information about legal places to break this occurrence of the word, should you need to break the word.

EXAMPLE

```
<p>Among the more common literary devices used by poets are
<hyphenate>(on\o\mat\o\poe\ia) and anthropomorphism.
```

The following shows the output this example may produce:

Among the more common literary devices used by poets are onomato-poeia and anthropomorphism.

<ICON>

Allows you to include a graphic image in your printed output and print text parallel to the image. The text is printed either to the right or left of the picture.

FORMAT

<ICON>

ARGUMENTS

None.

related tags

- <FIGURE_FILE>
- <ICON_FILE>
- <ICON_TEXT>

restrictions

Use the <ICON> tag for graphics that have a depth and width of approximately 2 inches. Use the <FIGURE_FILE> tag for larger graphics.

The output device must be able to support graphics.

required terminator

<ENDICON>

DESCRIPTION

Use the <ICON> tag when you want to print a small graphic with explanatory text printed next to it.

EXAMPLES

1

```
<ICON>
<ICON_FILE>(LN03\small_art.six\1.5\2.0\RIGHT)
<ICON_TEXT>(The text accompanying the
small piece of art. The text can be smaller or larger
than the graphic; the <tag>(ICON) tags make the necessary
adjustments for the output.)
<ENDICON>
```

This example shows how to code a graphic on the right, with text on the left.

2

```
<ICON>
<ICON_FILE>(LN03\SANTA.SIX\15\11)
<ICON_FILE>(LINE\Santa\3\8)
<ICON_TEXT>(The image at the left is of an American icon.
The personification of the spirit of Christmas,
usually represented as a jolly, fat old man with
a white beard and a red suit, is also called
<quote>(Saint Nicholas) or <quote>(Saint Nick.))
<ENDICON>
<ICON>
<ICON_FILE>(LN03\ELLIPSE.SIX\15\10\RIGHT)
<ICON_FILE>(LINE\ellipse\3\8\RIGHT)
<ICON_TEXT>(The image at the right is of an ellipse:
a conic section taken neither parallel to an element nor parallel
to the axis of the intersected cone.
<ENDICON>
```

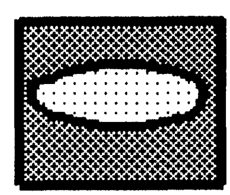
This example shows one graphic placed on the left, with text on the right, followed by a second graphic placed on the right, with text on the left. The output appears as follows:



ZK-7713-HC

The image at the left is of an American icon. The personification of the spirit of Christmas, usually represented as a jolly, fat old man with a white beard and a red suit, is also called "Saint Nicholas" or "Saint Nick."

The image at the right is of an ellipse: a conic section taken neither parallel to an element nor parallel to the axis of the intersected cone.



ZK-7714-HC

<ICON_FILE>

Specifies a graphics file that accompanies text within the <ICON> and <ENDICON> tags.

FORMAT

<ICON_FILE> *(target-device*
\ file-spec
\ vertical-size
\ horizontal-size
[\ RIGHT])

ARGUMENTS

target-device

Specifies a keyword indicating the output device for the graphics file. Keywords are provided both for devices that do (LN03 and PS) and do not (LINE, MAIL, and TERMINAL) support graphics. Each keyword allows you to insert the necessary amount of white space for the specific output device; keywords for devices that do not support graphics allow you to insert blank space in place of an icon. The following table lists the output devices and expected output for each keyword.

Keyword	Device	Output
LN03	LN03 Laser Printer	The specified sixel graphics file is output as an icon.
PS	PRINTSERVER 40 or LN03R SCRIPTPRINTER	The specified POSTSCRIPT graphics file[1] is output as an icon.
LINE MAIL TERMINAL	Line Printer	If specified, blank space is output with the file-spec argument written in that blank space. Only one of these three keywords should be used to indicate a monospaced destination.

[1]PostScript graphics files must conform to the Encapsulated PostScript File Format published by Adobe Systems Incorporated.

file-spec

Specifies the graphics file. No default file type is supplied.

vertical-size

Specifies the vertical size of the printed graphic in picas (there are 6 picas to an inch). This argument may be a nonnegative integer or decimal number, including zero.

horizontal-size

Specifies the width of the printed graphic in picas (there are 6 picas to an inch). This argument may be a nonnegative integer or decimal number, including zero.

RIGHT

Indicates that the graphic image is to be placed to the right of the text. If you do not specify this argument, the image is placed on the left of the text specified in the <ICON_TEXT> tag.

related tags

- <FIGURE_FILE>
- <ICON>
- <ICON_TEXT>

restrictions

You must have an output device that can print graphics files.

Only valid when used between the <ICON> and <ENDICON> tags.

required terminator

None.

DESCRIPTION

Specifies a graphics file that accompanies text within the <ICON> and <ENDICON> tags. The specified file should be a graphics file suitable for printing on the specified output device.

If you plan to process your file for more than a single destination, you can use multiple <ICON_FILE> tags to ensure that the appropriate icon will be included for the appropriate destination; an example of this coding is given in the examples in the <ICON> tag description. If you use multiple <ICON_FILE> tags in this way, you should specify only a single <ICON_FILE> tag for all the monospaced destination keywords (MAIL, TERMINAL, or LINE).

EXAMPLE

See the examples in the <ICON> tag description.

<ICON_TEXT>

Labels the text that accompanies a graphic image included in text with the <ICON> and <END_ICON> tags.

FORMAT

<ICON_TEXT> *(text)*

ARGUMENTS

text

Specifies the text that accompanies the graphic. It can be as long as you wish and can include paragraphs and lists.

related tags

- <ICON>
- <ICON_FILE>

restrictions

Must be used within the context of the <ICON> tag.

required terminator

None.

DESCRIPTION

The <ICON_TEXT> tag labels the text that accompanies a graphic image included in text with the <ICON> and <END_ICON> tags.

EXAMPLE

See the <ICON> tag description.

<INCLUDE>

Causes the contents of a specified file to be included in the current input file for processing.

FORMAT

<INCLUDE> *(file-spec)*

ARGUMENTS

file-spec

Specifies the file to be included.

If a logical name is specified, instead, and the source file is an element of a book, you can define the logical name using an <INCLUDES_FILE> tag in the book's profile. If the source file is not an element of a book, or if the profile does not contain the <INCLUDES_FILE> tag, be sure to define the logical name before processing the file through VAX DOCUMENT.

related tags

- <EXAMPLE_FILE>
- <TABLE_FILE>

restrictions

The <INCLUDE> tag is invalid in an argument to a tag.

In a file that contains a book element to be processed through a bookbuild, make sure to place the book element tag (for example, <CHAPTER>) as the first tag in the file. The <INCLUDE> tag should never be placed before the book element tag.

required terminator

None.

DESCRIPTION

The <INCLUDE> tag causes the contents of a specified file to be included in the current input file for processing.

EXAMPLE

```
<INCLUDE>(doc_local_templates:boilerplate.sdml)
```

This example shows the inclusion of a file that does the following:

- Contains text that might be repeated multiple times in a source file
- Is used in more than one document

If the file specification *doc_local_templates:boilerplate.sdml* was not used, but instead substituted with a logical name, the logical name could be equated to the file specification by using the <INCLUDES_FILE> tag in the profile.

<INCLUDES_FILE>

Equates a logical name with a file specification during processing of a profile.

FORMAT

<INCLUDES_FILE> *(logical-name \ file-spec)*

ARGUMENTS

logical-name

Specifies the logical name for the included file. You can use this name as the argument to an <INCLUDE> tag.

file-spec

Specifies the file specification into which the logical name translates.

related tags

- <PROFILE>
- <ELEMENT>
- <INCLUDE>

restrictions

Must be used in the context of a profile.

required terminator

None.

DESCRIPTION

You can place the <INCLUDES_FILE> tag in your profile to define a logical name for a file whose contents are included in one of your element files. The first argument to <INCLUDES_FILE> is the logical name for the included file. You use this name as the argument to the <INCLUDE> tag. The second argument specifies the actual file specification into which the logical name translates.

The <INCLUDES_FILE> tags follow the <ELEMENT> tag for which they are needed. During a bookbuild, VAX DOCUMENT establishes the logical name definition for each <INCLUDES_FILE> tag before it actually reads and processes the book element file names in the preceding <ELEMENT> tag. The logical name remains defined during the processing of later book elements.

When a book element is processed by itself (when you use the /PROFILE qualifier on the command line), VAX DOCUMENT again establishes the logical-name definitions that were specified by the <INCLUDES_FILE> tags in the profile.

EXAMPLE

```
<ELEMENT>(error_chap.SDML)
<INCLUDES_FILE>(Error_msg_tab\Mydisk:[Mydirectory]my_very_long_table.sdml)
```

This example illustrates how the <INCLUDES_FILE> tag is used to define a logical name for a file (called "my_very_long_table.sdml") which is being kept in the directory [Mydirectory]. In the book element file ERROR_CHAP.SDML, the writer includes the table of error message codes with the tag:

```
<include>(error_msg_tab).
```

<INDEX_FILE>

Specifies the position in a book (or document) where an index file should be included in the output.

FORMAT

<INDEX_FILE> *[(file-spec)]*

ARGUMENTS

file-spec

If you place the <INDEX_FILE> tag in an SDML file that will be included in another file later, you must specify the exact file-spec of the index as the <INDEX_FILE> tag's argument.

related tags

- <PROFILE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <INDEX_FILE> tag specifies the position in a source file where an index output file should be included. This tag does not produce an index, but simply indicates placement of the index file. An index file is produced when the qualifier "/INDEX" is specified on the DOCUMENT command line.

The profile tags <INDEX_FILE> and <CONTENTS_FILE> can be placed in either the profile of a book or in a source file. If you are creating a book (with a profile to be processed through a bookbuild), place these tags in the profile to be sure of correct placement in the output.

An index file always receives the filetype .DVI *_device*, where *device* is the type of output device you specified on the command line. For more information on index generation, see Chapter 7.

To create an index from an individual file that contains an <INDEX_FILE> tag, specify the "/INDEX" qualifier on the command line.

EXAMPLE

To see the result of the <INDEX_FILE> tag, refer to the index in this manual. The <INDEX_FILE> tag was placed in the profile file after the last appendix file.

<INTERACTIVE>

Begins an example dialog between user and system and enables the tags <S> and <U> to distinguish system text from user text.

FORMAT

<INTERACTIVE> *(code)*

or

<INTERACTIVE> *[([KEEP \ ] [WIDE[\ MAXIMUM]])]*

interactive code or text

.

.

.

<ENDINTERACTIVE>

ARGUMENTS

code

Specifies a code fragment you want to insert into your text.

If this argument is not specified, the terminator <ENDINTERACTIVE> is required.

KEEP

Specifies that the example is not to be broken across pages, that is, if the example does not fit on the current page, it will be placed on the next page. If the example itself does not fit on a single page of output, it will be broken anyway.

WIDE

Specifies that the width of the example exceeds the document's default width for text. Depending on the document type, this argument can be interpreted as follows:

- If the document style contains a left margin area that is normally used for headings, the example's width will span that area as well as the normal text area.
- If the document uses a multicolumn format, multicolumn output is suspended while the example is processed. The example is output and multicolumn output is then restored.
- If the document style provides a range of sizes and styles for examples, this argument may be interpreted to mean that a specific size should be used for the example.

MAXIMUM

Can be used in conjunction with WIDE to indicate that the example may require additional adjustment to fit within the bounds of the text page. This argument must be used with discretion, and may not be suitable in all document styles.

related tags

- <S>
- <U>
- <VALID_BREAK>

restrictions

Indexing tags (<X> and <Y> tags) are not permitted within interactive examples.

Tab characters cannot be used to format interactive examples; use spaces instead.

Do not use text element tags within interactive examples (for example, <P> , <LIST> , or <NOTE>).

required terminator

<ENDINTERACTIVE>

DESCRIPTION

The <INTERACTIVE> tag indicates the beginning of an example dialog between user and system. It enables the tags <S> and <U> . If your interactive example is longer than a few lines, use the <VALID_BREAK> tag to indicate the acceptable points for a page break within that example.

EXAMPLE

```
<P>If you do not specify the full command line, DCL will prompt
you for the missing information. For example, if you do not specify an
input file and an output file when you enter the COPY command, you will
be prompted as follows:
<INTERACTIVE>
<S>($ )<U>(COPY)
<S>($_From:        )<U>(INTERACT.GNC)
<S>($_To:          )<U>(NEWFILE.GNC)
<ENDINTERACTIVE>
```

Note that you should specify whatever space follows the system prompt within the <S> tag. This example may produce the following output:

If you do not specify the full command line, DCL will prompt you for the missing information. For example, if you do not specify an input file and an output file when you enter the COPY command, you will be prompted as follows:

```
$ COPY
$_From:        INTERACT.GNC
$_To:          NEWFILE.GNC
```

<KEEP>

Specifies that a string of text should always occur on the same line of output.

FORMAT

<KEEP> *(text)*

ARGUMENTS

text
Specifies the text string to be kept on the same line of output.

related tags

- The following tags are used in conjuction with <KEEP> to allow a user to specify formatting attributes:

  <EMPHASIS>
  <DEFINE_SYMBOL>
  <HYPHENATE>

restrictions

None.

required terminator

None.

DESCRIPTION

Specifies that a string of text should always occur on the same line of output; that is, it should not be broken between lines. The <KEEP> tag is used either to prevent hyphenation of a word or to prevent a string of text from breaking across a line.

EXAMPLES

1

```
<P>
The complete file specification is: <KEEP>(DISK$:[SMITH.TRIPS.EXPENSES]MILEAGE.TXT)
```

In this example, the file specification is specified as an argument to the keep tag so that it will not be broken across a line. This code fragment may produce the following output:

The complete file specification is:
DISK$:[SMITH.TRIPS.EXPENSES]MILEAGE.TXT

2

```
<define_symbol>(VAX11\<delayed><keep>(VAX--11)<enddelayed>)
```

This defines a text element symbol-name using an en dash and specifies that the line should not break between the en dash and the 11 in the reference <REFERENCE> (vax11).

A reference to this defined symbol would produce the following output:

VAX–11

<KEYWORD>

Labels a significant word that deserves to be distinguished typographically.

FORMAT

<KEYWORD> *(word)*

ARGUMENTS

word

Specifies the word to be distinguished.

related tags

- <VARIABLE>
- <SPECIAL_NAME>

restrictions

None.

required terminator

None.

DESCRIPTION

The <KEYWORD> tag labels a word or term that you want to distinguish typographically. The default action of the <KEYWORD> tag outputs the keyword in boldface. From the point of view of formatted result, the <KEYWORD> tag may appear similar to the <EMPHASIS> tag with the BOLD argument. However, by using separate tags to label different kinds of information, the book designer is free to change the format of any one kind without affecting the others.

What constitutes a keyword is something about which both editor and writer must agree. The presence of keywords must be taken into account in the book design. Use of the <KEYWORD> tag should be consistent within a document and across a document set.

EXAMPLE

```
<P>A <KEYWORD>(field) is a set of contiguous bytes in a logical record.
```

This example may produce the following output:

A **field** is a set of contiguous bytes in a logical record.

<LE>

Labels a list element.

FORMAT

<LE> *[(callout-number)]*

ARGUMENTS

callout-number

Used only within the context of <LIST> (CALLOUT) to set the callout number identifying the list element. See the examples in the discussion of <LIST> (CALLOUT).

related tags

- <LIST>

restrictions

Must be used within the context of the <LIST> tag.

required terminator

None.

DESCRIPTION

The <LE> tag identifies a list element and is only valid when used within the context of the <LIST> tag.

EXAMPLES

See the examples in the discussion of the <LIST> tag.

<LINE>

Specifies that the text that follows is to be placed on a new line of output.

FORMAT

<LINE> *[([INDENT[\ unit-number] | [BIGSKIP | SMALLSKIP]])]*

ARGUMENTS

INDENT

Specifies that the next line or block of text is to be indented from the preceding text. If the INDENT argument is specified, the next argument must be an integer from 1 to 9 indicating the number of units that the text is to be indented. The size of these units is determined separately for each document design.

The default indent is 1 unit; the maximum indent is 9 units.

BIGSKIP
SMALLSKIP

Specifies that a set amount of vertical space is to precede the element identified as a line or block of text. The actual amount of space created is determined by the document's design.

related tags

- <FINAL_CLEANUP>
- <CENTER_LINE>
- <RIGHT_LINE>

restrictions

Invalid in monospaced examples and math.

required terminator

None.

DESCRIPTION

The result produced by this tag differs according to whether it is used in the context of a <P> tag, a <TABLE_ROW> tag, or a <FORMAT> tag:

- When you specify <LINE> in the context of a paragraph of text or a list element, the <LINE> tag causes the current paragraph to be terminated and then starts a new block of text. By default, the text block is not preceded by any extra vertical space nor is it indented.
- When you specify the <LINE> tag in the context of a <TABLE_ROW> or a <FORMAT> tag, the <LINE> tag causes the next text to begin on a new line of output, but does not modify the current paragraph.

Do not use the <LINE> tag in a paragraph to cause the text formatter to break a line within the paragraph. Breaking a line to override a specific line break is a final formatting/cleanup instruction and you should use the <FINAL_CLEANUP> (LINE_BREAK) tag for this.

EXAMPLES

1
```
<LIST>(NUMBERED)
<le>ITEM
<LINE>This item specifies...
```

The output may be formatted as follows:

1 ITEM
This item specifies . . .

2
```
<P>This is a normal paragraph.
<line>(INDENT\1\SMALLSKIP)
This is a block paragraph, indented with skip.
```

This may produce:

This is a normal paragraph.

This is a block paragraph, indented with skip.

3
```
<P>Flashy designs are inappropriate for software manuals or for
any serious or formal books because they do not reflect the
intention of the writer. The purpose of any book design is
<LINE>
to clarify what the author is conveying, to translate the text
attractively as print on a page, to communicate the message
visually in harmony with the ideas.
```

This output illustrates misuse of the <LINE> tag for formatting within a paragraph. Although correct results may be obtained on a given output device for a particular run of a file, the output may also be formatted as follows:

Flashy designs are inappropriate for software manuals or for any serious or formal books because they do not reflect the intention of the writer. The purpose of any book design is

to clarify what the author is conveying, to translate the text attractively as print on a page, to communicate the message visually in harmony with the ideas.

4
```
<TABLE>(Card Reader Errors: Causes and Corrective Actions\cardread_tab)
<TABLE_ATTRIBUTES>(wide\multipage)
<TABLE_SETUP>(3\12\21)
<TABLE_HEADS>(Error\Causes\Corrective Action)
<TABLE_ROW>(READ CHECK\Card edges torn <LINE> Punch in column 0
or 81\Remove the faulty card from the output stacker, duplicate the card, place
it in the input hopper, and press the <EMPHASIS>(RESET\bold) button.)
<TABLE_ROW_BREAK>(FIRST)
<TABLE_ROW>(PICK CHECK\Damage to leading edge <LINE> Torn webs
<LINE> Cards
stapled together\Remove the card from the input hopper, duplicate
the faulty card, place the card back in the input
hopper, and press the <EMPHASIS>(RESET\bold) button.)
<TABLE_ROW_BREAK>(LAST)
<TABLE_ROW>(STACK CHECK\Jam in the card track <LINE> Badly mutilated card
\Correct the jam and/or remove the mutilated card from the output stacker,
duplicate the card, place it in the input hopper, and press the
<EMPHASIS>(RESET\bold) button.)
<TABLE_ROW>(HOPPER CHECK\Input hopper empty <LINE> Output stacker full\Load
the input hopper. <LINE> Unload the output stacker.)
<ENDTABLE>
```

This example shows the use of the <LINE> tag in a three-column table. It may produce the following output:

Table x—x Card Reader Errors: Causes and Corrective Actions

Error	Causes	Corrective Action
READ CHECK	Card edges torn Punch in column 0 or 81	Remove the faulty card from the output stacker, duplicate the card, place it in the input hopper, and press the **RESET** button.
PICK CHECK	Damage to leading edge Torn webs Cards stapled together	Remove the card from the input hopper, duplicate the faulty card, place the card back in the input hopper, and press the **RESET** button.
STACK CHECK	Jam in the card track Badly mutilated card	Correct the jam and/or remove the mutilated card from the output stacker, duplicate the card, place it in the input hopper, and press the **RESET** button.
HOPPER CHECK	Input hopper empty Output stacker full	Load the input hopper. Unload the output stacker.

5
```
<FORMAT>(MY COMMAND)
<FCMD>( )
<FPARMS>(one parameter<line>two<line>three)
<ENDFORMAT>
```

This example shows the <LINE> tag in a <FORMAT> tag. This example may produce the following output:

MY COMMAND *one parameter*
two
three

<LINE_ART>

Labels a rough sketch produced at the terminal keyboard for draft output, to give some idea of what the final figure will look like.

FORMAT

<LINE_ART> *[([KEEP]*
[WIDE[\ MAXIMUM]])]

ARGUMENTS

KEEP

Specifies that the line art is not to be broken across pages, that is, if the line art does not fit on the current page, it will be placed on the next page. If the line art itself does not fit on a single page of output, it will be broken anyway.

WIDE

Specifies that the width of the line art exceeds the document's default width for text. Depending on the document type, this argument can be interpreted as follows:

- If the document style contains a left margin area that is normally used for headings, the line art's width will span that area as well as the normal text area.
- If the document uses a multicolumn format, the line art will result in suspension of multicolumn output while it is being processed. The line art will be output, and then multicolumn output will be restored.
- If the document style provides a range of sizes and styles for examples, this argument may be interpreted to mean that a specific size should be used for the line art.

MAXIMUM

Can be used in conjunction with WIDE to indicate that the line art might require additional adjustment to fit within the bounds of the text page. This argument must be used with discretion, and may not be suitable in all document styles.

related tags

- <FIGURE>
- <VALID_BREAK>

restrictions

None.

required terminator

<ENDLINE_ART>

<LINE_ART>

DESCRIPTION

The <LINE_ART> tag labels a rough sketch produced at the terminal keyboard for draft output, to give some idea of what the final figure will look like.

Note that the results of keyboard drawing may be adequate for draft purposes, but will not be acceptable in the context of laser printer output.

EXAMPLE

See the examples in the discussion of the <FIGURE> tag.

<LIST>

Begins a list. The type of list (for example, numbered or stacked) is specified by the argument to the <LIST> tag.

FORMAT

<LIST> *(keyword[\ attributes])*

ARGUMENTS

KEYWORD

Specifies the type of list.

ALPHABETIC	The list element identifiers are alphabetic letters.
CALLOUT	The list element identifiers are reverse-print callout numbers (on supported output devices), for example ❺.
NUMBERED	The list element identifiers are Arabic numerals.
ROMAN	The list element identifiers are Roman numerals.
SIMPLE	There are no list element identifiers.
STACKED	Individual list elements do not have identifiers, but the entire list is stacked within the specified set of delimiters (braces, brackets, double brackets, or single or double vertical rules.)
UNNUMBERED	List element identifiers are special characters.

ATTRIBUTES

Specifies the attributes of the list element identifiers. More than one attribute can be specified.

Start-letter	For an ALPHABETIC list. Specifies the alphabetic letter to use for the first item in the sequence. Subsequent items are automatically incremented.
Uppercase	For an ALPHABETIC or ROMAN list. Specifies that the list element identifiers (alphabetic letters or Roman numerals) are to be printed in uppercase letters. By default, Alphabetic and Roman numeral list element identifiers are printed using lowercase letters.
Start-number	For a NUMBERED or ROMAN list. Specifies the number to be assigned to the first list element. Subsequent list elements are automatically incremented.
BRACES BRACKETS DOUBLE_BRACKETS VERTICAL_RULE DOUBLE_VERTICAL_RULE	For a STACKED list. Each attribute for a STACKED list specifies the delimiter to be used to surround the stacked elements. If no keyword is specified, list elements are stacked without a surrounding delimiter.

Char	For an UNNUMBERED list. Specifies a single character or a tag that results in a single printed character of output that will be used to indicate list elements. If *char* is not specified, the bullet character is used.

related tags

- <LE>

restrictions

The <LIST> tag should only used in the context of a paragraph or table.

<LIST> tags that use the following keywords may be nested (that is, an element of a list may itself contain the beginning of another list): ALPHABETIC, NUMBERED, UNNUMBERED, ROMAN, or SIMPLE. However, none of these list types are compatible with <LIST> (STACKED). Stacked lists may only be nested within other stacked lists, and stacked lists may not be nested within any of the previously noted lists.

required terminator

<ENDLIST>

DESCRIPTION

The <LIST> (ALPHABETIC), <LIST> (NUMBERED), and <LIST> (ROMAN) tags begin a list whose elements have a particular sequence or priority. Use the UNNUMBERED or SIMPLE arguments to begin a list that has no inherent order or priority.

Alphabetic lists are useful nested within numbered lists. A numbered list is used to indicate a particular sequence or priority within the list elements.

The SIMPLE argument labels a simple list with no enumerator or special character preceding each list element.

The STACKED argument begins a list whose elements are left-justified on successive lines of an imaginary box. The box can have large braces or brackets placed on each side. The box is then centered vertically so that it aligns with text to the left or right on the same line. (By contrast, <LIST> (SIMPLE) terminates the current paragraph and indents the list, so that it is seen as a separate entity from the text above and below it.) <LIST> (STACKED) is especially useful for showing syntactic elements with a <FORMAT> tag.

The UNNUMBERED argument labels lists that have no particular order or priority within the list element. Normally a bullet is the character used before each list element. You can specify another character by passing it as an argument.

EXAMPLES

1

```
<LIST>(NUMBERED)
<LE>Review doc plan
<LE>Find out these numbers:
     <LIST>(UNNUMBERED)
     <LE>LPN
```

```
     <LE>DPN
     <LE>order number
     <ENDLIST>
     <LE>For revisions of books not already in the library, have
production or the writer rename  the file
according to its new LPN.
     <LE>Open a library for the book if none exists (except for
one-shot jobs).
     <LIST>(ALPHABETIC)
     <LE>Decide with writer about timing:
          <LIST>(UNNUMBERED)
          <LE>Must be done before final production.
          <LE>Shouldn't be done until text is stable.
          <LE>Preferred time is at major edit pass.
          <ENDLIST>
     <LE>Cooperate with production librarian on paperwork.
     <LE>Help writers name new element files correctly.
     <LE>Verify that all files are of the same file type.
     <LE>For a revision, be sure files are renamed using the
     revision's LPN when the library is created for the revision.
     <ENDLIST>
<ENDLIST>
```

In this example, the first nested list has no particular sequence. The second nested list does have an order to it and this order is reflected in the use of the <LIST> (alphabetic) tag.

This example may produce the following output:

1. Review doc plan
2. Find out these numbers:
   - LPN
   - DPN
   - order number
3. For revisions of books not already in the library, have production or the writer rename the file according to its new LPN.
4. Open a library for the book if none exists (except for one-shot jobs).
   - **a.** Decide with writer about timing:
     - Must be done before final production.
     - Shouldn't be done until text is stable.
     - Preferred time is at major edit pass.
   - **b.** Cooperate with production librarian on paperwork.
   - **c.** Help writers name new element files correctly.
   - **d.** Verify that all files are of the same file type.
   - **e.** For a revision, be sure files are renamed using the revision's LPN when the library is created for the revision.

2

```
<P>Items e and f describe the $GETJPI AST activity:
<LIST>(ALPHABETIC\5)
<LE>An ACB is constructed for a special kernel AST.
<LE>When the special kernel mode AST routine executes in the context of
the target process, the requested information is moved into the system buffer.
The ACB is then reset to deliver a special kernel mode AST back to the
requesting process.
<ENDLIST>
```

This example may produce the following output:

Items e and f describe the $GETJPI AST activity:

e. An ACB is constructed for a special kernel AST.

f. When the special kernel mode AST routine executes in the context of the target process, the requested information is moved into the system buffer. The ACB is then reset to deliver a special kernel mode AST back to the requesting process.

3

```
<P>At this point, you can log in to the system as the system manager
by performing the following steps at the console terminal:
<LIST>(NUMBERED)
<LE>Press RETURN.
<LE>In response to the system's request for your user name, type SYSTEM.
<LE>In response to the system's request for your password, type MANAGER.
<ENDLIST>
```

This example produces the following output:

At this point, you can log in to the system as the system manager by performing the following steps at the console terminal:

1 Press RETURN.

2 In response to the system's request for your user name, type SYSTEM.

3 In response to the system's request for your password, type MANAGER.

4

```
<P>Items six and seven describe two main principles of a generic
markup language:
<LIST>(NUMBERED\6)
<LE>Descriptive markup predominates and is distinguished from processing
instructions.
<LE>Markup is formally defined for each type of document.
<ENDLIST>
```

This example has the following output:

Items six and seven describe two main principles of a generic markup language:

6 Descriptive markup predominates and is distinguished from processing instructions.

7 Markup is formally defined for each type of document.

5

```
<P>The following items are needed:
<LIST>(SIMPLE)
<LE>bread
<LE>milk
<LE>cheese
<LE>cereal
<LE>fruit
<ENDLIST>
```

This example produces the following output:

The following items are needed:

bread
milk
cheese
cereal
fruit

6

```
<P>
ON <LIST>(stacked\braces)
      <LE>ANYCONDITION
      <LE>ENDFILE(reference)
      <LE>ENDPAGE(reference)
      <LE>FINISH
      <LE>KEY(reference)
      <LE>UNDEFINEDFILE(reference)
      <LE>ERROR
      <LE>FIXEDOVERFLOW
      <LE>OVERFLOW
      <LE>UNDERFLOW
      <LE>VAXCONDITION(expression)
      <LE>ZERODIVIDE
   <ENDLIST>

   <LIST>(stacked\braces)
      <LE>statement
      <LE>begin-block
   <ENDLIST>
```

Notice that the tags are indented to help in visually checking the matching of the <LIST> (stacked) and <ENDLIST> tags. This indentation has no effect on the output, however. This example may produce the following:

ON { ANYCONDITION
ENDFILE(reference)
ENDPAGE(reference)
FINISH
KEY(reference)
UNDEFINEDFILE(reference)
ERROR
FIXEDOVERFLOW
OVERFLOW
UNDERFLOW
VAXCONDITION(expression)
ZERODIVIDE } { statement
begin-block }

7
```
<P>
OPEN FILE (reference) [TITLE(expression)]
<P>
<LIST>(stacked\brackets)
       <LE>[STREAM] <LIST>(stacked\brackets)
                      <LE>[INPUT]
                      <LE>OUTPUT [LINESIZE] [PRINT [PAGESIZE(integer)] ]
             <ENDLIST>
       <LE>RECORD <LIST>(stacked\brackets)
                      <LE>[INPUT]
                      <LE>OUTPUT
                      <LE>UPDATE
                  <ENDLIST>
                          <LIST>(stacked\brackets)
                             <LE>DIRECT
                             <LE>[SEQ[UENTIAL]]
                          <ENDLIST>
                           [KEYED]
        <ENDLIST>
```

This example may produce the following:

OPEN FILE (reference) [TITLE(expression)]

[STREAM] [INPUT]
OUTPUT [LINESIZE] [PRINT [PAGESIZE(integer)]]

RECORD [INPUT]
OUTPUT
UPDATE [DIRECT
[SEQ[UENTIAL]]] [KEYED]

8
```
<TABLE>
<TABLE_SETUP>(2\20)
<TABLE_HEADS>(Some stacked\Stuff in a Table)
<TABLE_ROW>(<LIST>(stacked\braces)
           <LE>one
           <LE>two
           <LE>three
          <ENDLIST>\That was stacked in a table.)
<TABLE_ROW>(Just a second item\<LIST>(stacked\braces)
                                 <LE>one
                                 <LE>two
                                 <LE>three
                                <ENDLIST>)
<TABLE_ROW>(this is the last item)
<ENDTABLE>
```

Here, <LIST> (stacked) arguments are used within a table. Again, the generic code is indented simply as an aid for visually checking the nesting of tags. This example may produce the following:

Some stacked	**Stuff in a Table**
{ one two three }	That was stacked in a table.
Just a second item	{ one two three }
this is the last item	

9
```
<P>To create a system that more closely suits the requirements of
your site, you can do any of the following:
<LIST>(UNNUMBERED)
<LE>Select a default bootstrap command procedure
<LE>Modify system parameters for special hardware configuration needs or
special workload requirements
<LE>Perform other site-specific modifications
<ENDLIST>
```

This example may produce the following output:

To create a system that more closely suits the requirements of your site, you can do any of the following:

- Select a default bootstrap command procedure
- Modify system parameters for special hardware configuration needs or special workload requirements
- Perform other site-specific modifications

10
```
<P>To create a system that more closely suits the requirements of
your site, you can do any of the following:
<LIST>(UNNUMBERED\+)
<LE>Select a default bootstrap command procedure
<LE>Modify system parameters for special hardware configuration needs or
special workload requirements
<LE>Perform other site-specific modifications
<ENDLIST>
```

This example shows how you can specify a character other than a bullet to label each list element. This example may produce the following output:

To create a system that more closely suits the requirements of your site, you can do any of the following:

+ Select a default bootstrap command procedure
+ Modify system parameters for special hardware configuration needs or special workload requirements
+ Perform other site-specific modifications

<LITERAL>

Allows you to specify text that contains words in angle brackets that might otherwise be interpreted as tags.

FORMAT

<LITERAL> *(literal-text)*

or

<LITERAL>
literal-text

.
.
.

<ENDLITERAL>

ARGUMENTS

literal-text
The literal text to be passed through the tag translator unprocessed.

related tags

None.

restrictions

None.

required terminator

<ENDLITERAL> —Required if the text is not specified as an argument.

EXAMPLE

```
<P> I've experimented with my own method for noting questionable
phrases, by placing <LITERAL><NOTE_PROBLEM><ENDLITERAL> near the
phrase.
```

This example may have the following output:

I've experimented with my own method for noting questionable phrases, by placing <NOTE_PROBLEM> near the phrase.

<LOWERCASE>

Labels text that should appear as lowercase in the final output.

FORMAT

<LOWERCASE> *(text)*

ARGUMENTS

text
Specifies the text to appear in lowercase.

related tags

- <UPPERCASE>

restrictions

None.

required terminator

None.

DESCRIPTION

In your book, there may be a text element, such as a heading, that normally appears in uppercase. For example, in some book designs, first-level headings use all uppercase letters.

You may encounter a situation where you need to overcome the default case in one of your tags and ensure that the result in the final output appears in lowercase. The <LOWERCASE> tag allows you to do this.

EXAMPLE

```
<HEAD2>(HeRe iS aN ExAmPlE oF <LOWERCASE>(lOwErCaSe) tExT)
```

In this example, assume that the doctype being used causes the tag <HEAD2> to output a heading that is in uppercase, no matter what the case of the text passed to it.

The example may produce the following output:

HERE IS AN EXAMPLE OF lowercase TEXT

The default to uppercase letters causes the heading to be uniformly uppercase, with the exception of the text passed to the <LOWERCASE> tag.

<MARK>

Indicates the beginning of new or modified information.

FORMAT

<MARK>

ARGUMENTS

None.

related tags

- <REVISION>
- <UPDATE_RANGE>

restrictions

Output is enabled only if the <REVISION> tag has been specified.

required terminator

<ENDMARK>

DESCRIPTION

The <MARK> and <ENDMARK> tags delimit a sequence of text that has been modified. These tags will produce vertical bars in the margin of the document.

Make sure to place the <MARK> and <ENDMARK> tags next to the text they mark, without preceding or following the text with other tags. The text formatter interprets <MARK> based on the last text character encountered before the <MARK> tag.

EXAMPLE

```
<P>
The following characters are legal in MACRO-11 source programs:
<LIST>(UNNUMBERED)
<LE>The letters A through Z. Both upper- and lowercase letters are
acceptable, although, upon input, lowercase  letters are converted to
uppercase.
<LE>The digits 0 through 9.
<LE>The characters period <PARENDCHAR>(.) and dollar sign
<PARENDCHAR>($). These characters  are reserved for use as Digital
Equipment Corporation system program symbols.
<ENDLIST>
```

```
<REVISION>
    .
    .
    .
<P>
The following characters are legal in MACRO-11 source programs:
<LIST>(UNNUMBERED)
<LE>The letters A through Z. Both upper- and lowercase letters are
acceptable, although, upon input, lowercase  letters are converted to
uppercase.

<MARK>
<LE>Characters in the DEC multinational character set (MCS). A chart
showing the MCS is located in <REFERENCE>(mcs_app),
with a list of directives that
support the MCS.
<ENDMARK>

<LE>The digits 0 through 9.
<LE>The characters period <PARENDCHAR>(.) and dollar sign
<PARENDCHAR>($). These characters  are reserved for use as Digital
Equipment Corporation system program symbols.
<ENDLIST>
```

In this example, the first paragraph is the original and the second paragraph is the modified version with the <MARK> and <ENDMARK> tags coded in. The formatted output for the original paragraph and the modified paragraph may be as follows:

The following characters are legal in MACRO–11 source programs:

- The letters A through Z. Both upper- and lowercase letters are acceptable, although, upon input, lowercase letters are converted to uppercase.
- The digits 0 through 9.
- The characters period (.) and dollar sign ($). These characters are reserved for use as Digital Equipment Corporation system program symbols.

The following characters are legal in MACRO–11 source programs:

- The letters A through Z. Both upper- and lowercase letters are acceptable, although, upon input, lowercase letters are converted to uppercase.
- Characters in the DEC multinational character set (MCS). A chart showing the MCS is located in Appendix A, with a list of directives that support the MCS.
- The digits 0 through 9.
- The characters period (.) and dollar sign ($). These characters are reserved for use as Digital Equipment Corporation system program symbols.

<MATH>

Labels a short mathematical expression or the beginning of an extended mathematical example.

FORMAT

<MATH> *(* { *math-expression* | **DISPLAY***[* \ *symbol-name]* } *)*

ARGUMENTS

math-expression

Specifies a mathematical expression to be included in the text of a sentence or paragraph.

DISPLAY

Specifies a keyword indicating that an extended mathematical equation or expression is to be set off from the surrounding text.

symbol-name

Specifies a symbol-name by which the extended mathematical equation is to be referenced. If this argument is specified, the equation is assigned a number and the number is printed to the right of the equation.

Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

Once the symbol-name is defined, the equation can be referenced using the <REFERENCE> tag.

related tags

- <MATH_CHAR>
- <CODE_EXAMPLE>
- <REFERENCE>

restrictions

- The use of extended mathematical examples is not valid within tables or monospaced examples.
- The set of tags valid within a mathematical expression is limited to those listed in the tables in the following "Description" section.

required terminator

<ENDMATH> —Required when DISPLAY is specified as an argument to <MATH> .

DESCRIPTION

Within an argument to <MATH> or within the bounds of <MATH> (DISPLAY) . . . <ENDMATH> , you can specify simple or complex mathematical expressions, according to the rules outlined below. The context in which you enter text and tags within a mathematical expression is severely restricted.

In a mathematical expression, all formatting is controlled by the text processor. When you use the keyword DISPLAY, the output will be offset from the surrounding text, but you do not have any control over its positioning. Furthermore, blank spaces and carriage returns are ignored; the text processing program assumes that all text strings are mathematical variable names, and makes all decisions regarding the output formatting.

Simple Expressions

The following tags are used to indicate simple binary operations. In the context of a mathematical expression, you can use either the tag name or the function's symbol.

<TIMES>	*	Multiplication
<PLUS>	+	Plus
<MINUS>	-	Minus
<DIVIDED_BY>	/	Division
<EQUALS>	=	Equality

For example, the following are equivalent:

```
<MATH>(total = A + B - C * D / E)
<MATH>(total <EQUALS> A <PLUS> B <MINUS> C <TIMES> D <DIVIDED_BY> E)
```

Both produce the output: $total = A + B - C * D/E$

Note: The use of any nonalphanumeric characters other than those specified above in the context of math may produce unpredictable results, including errors from the text formatter.

Operators are evaluated using the normal mathematical rules for precedence of operators. You can control the evaluation by using the <GROUP> tag, much as you would use parentheses in a mathematical expression. Note the use of the <GROUP> tag in the second math expression in the following code fragment:

```
<MATH>(total = A +B-C*D<OVER>(e) )
<P>
<MATH>(total = <GROUP>(<GROUP>(A +B-C*D)<OVER>(e)) )
```

This code fragment may produce the following output:

$\frac{total{=}A{+}B-C*D}{e}$

$total = \frac{A{+}B-C*D}{e}$

The effect of the <GROUP> tag is supplied by default for fractional expressions when you use the <FRACTION> tag:

```
<MATH>(display)
total = <FRACTION>(A + B - C * D\E)
<ENDMATH>
```

Variable Names

The text formatter assumes that variable names are primarily alphabetic letters or special characters represented by the <MATH_CHAR> tag. If you need to specify any variable names in expressions that contain special characters, you must use the <VARIABLE> or <TEXT> tags. For example:

```
<MATH>(<variable>(event_flag) = 1)
```

If you require multiword variable names, you must use the <SP> tag to indicate spacing:

```
<MATH>(display)
<VARIABLE>(SUCCESS<sp>RATE) <equals>
<group>(
<VARIABLE>(TOTAL<SP>HITS) <over>(<VARIABLE>(TOTAL<sp>HITS) <plus>
                              <VARIABLE>(TOTAL<sp> MISSES)
                                )
       )
<times> 100
<ENDMATH>
```

This example may produce the following output:

$$SUCCESS\ RATE = \frac{TOTAL\ HITS}{TOTAL\ HITS + TOTAL\ MISSES} * 100$$

You can provide special annotation for variable names by using the following tags:

Tag	Output
<BAR_CHAR> (x)	$\bar{x}$
<DOT_CHAR> (x)	$\dot{x}$
<HAT> (xyz)	$\widehat{xyz}$
<HAT_CHAR> (x)	$\hat{x}$
<OVERLINE> (var)	$\overline{var}$
<TILDE> (xyz)	$\widetilde{xyz}$
<TILDE_CHAR> (x)	$\tilde{x}$
<UNDERLINE> (var)	$\underline{var}$
<VECTOR> (x)	$\vec{x}$

Parentheses

Most of the tags valid in math do not accept arguments. However, parentheses are frequently used in mathematical expressions. You must be careful, therefore, to use spaces preceding the parentheses. For example:

```
<MATH>(A <minus> (B<PLUS>C))
```

Summary of Tags

Table 9–3 summarizes the tags that are valid in mathematical expressions.

Table 9–3 <MATH> Expressions

Tag	Operation and Output
<AMPERSAND>	Specifies a literal ampersand character in a math expression.
<ATOP> (*expression*)	Stacks an expression, as in ${a \atop b}$.
<BACKSLASH>	Specifies a literal backslash character in a math expression.
<CAL> (*letter*)	Specifies a calligraphic uppercase letter, as in $\mathcal{A}$.
<CASES> <CASE_ROW> <ENDCASES>	Specifies a case construction. See "Matrices and Cases."
<CDOTS>	Specifies centered dots, as in $\cdots$.
<CHOOSE> (*expression1\expression2*)	Parenthetical notation, as in $\binom{a}{b}$.
<DDOTS>	Specifies diagonal dots, as in $\ddots$.
<DIVIDED_BY>	Division, as in a/b.
<DOTS>	Specifies horizontal dots, as in $\ldots$.
<DOT_TIMES>	Multiplication, as in $a \cdot b$.
<EQUALS>	Equality, as in $a = b$.
<FRACTION> (*numerator\denominator*)	Specifies a fraction, as in $\frac{a}{b}$.
<FUNC> (*expression*)	Specifies a function, as in $f(a)$.
<GROUP> (*expression*)	Provides control over the order of operation. See "Examples," in the following section.
<INTEGRAL>	Specifies an integral, as in $\int_0^{\frac{\pi}{2}}$. See "Operators With and Without Limits."
<INTEGRAL_LIMITS>	Specifies an integral that places superscripts and subscripts above and below the sign, rather than to the right. See "Operators With and Without Limits."
<INTEGRAL_NOLIMITS>	Specifies an integral that places superscripts and subscripts to the right of the sign, rather than above and below. See "Operators With and Without Limits."
<MATRIX> <MATRIX_ROW> <ENDMATRIX>	Specifies a matrix. See "Matrices and Cases."
<MINUS>	Minus, as in $a - b$.
<MOD>	Modulo, as in $n \bmod p$.

Table 9–3 (Cont.) <MATH> Expressions

Tag	Operation and Output
<LBAR>	Begins an expression delimited with vertical bars. See "Delimited Expressions."
<LBRACE>	Begins an expression delimited with curly braces. See "Delimited Expressions."
<LBRACKET>	Begins an expression delimited with square brackets. See "Delimited Expressions."
<LCEIL>	Begins an expression delimited with ceil characters ($\lceil$). See "Delimited Expressions."
<LFLOOR>	Begins an expression delimited with floor characters ($\lfloor$). See "Delimited Expressions."
<LPAREN>	Begins an expression delimited with parentheses. See "Delimited Expressions."
<OVER> *(expression)*	Division, as in $\frac{a}{b}$.
<PI>	The pi character, π.
<PLUS>	Plus, as in $a + b$.
<PMOD> *(expression)*	Parenthetical mod, as in $a \pmod p$.
<PROD>	Specifies a product, as in $\prod_0^{\frac{\pi}{2}}$. See "Operators With and Without Limits."
<PROD_LIMITS>	Specifies a product that places superscripts and subscripts above and below the sign, rather than to the right. See "Operators With and Without Limits."
<PROD_NOLIMITS>	Specifies a product that places superscripts and subscripts to the right of the sign, rather than above and below. See "Operators With and Without Limits."
<RBAR>	Ends an expression delimited with vertical bars. See "Delimited Expressions."
<RBRACE>	Ends an expression delimited with curly braces. See "Delimited Expressions."
<RBRACKET>	Ends an expression delimited with square brackets. See "Delimited Expressions."

Table 9–3 (Cont.) <MATH> Expressions

Tag	Operation and Output
<RCEIL>	Ends an expression delimited with ceiling characters ($\rceil$). See "Delimited Expressions."
<RFLOOR>	Ends an expression delimited with floor characters ($\rfloor$). See "Delimited Expressions."
<RPAREN>	Ends an expression delimited with parentheses. See "Delimited Expressions."
<SP>	Provides space in a variable name or expression.
<SQRT> (*expression*)	The square root, as in $\sqrt{a}$.
<SUBSCRIPT> (*expression*)	Subscription, as in a_b.
<SUM>	Specifies summation, as in $a \sum b$. See "Operators With and Without Limits."
<SUM_LIMITS>	Specifies summation that places superscripts and subscripts above and below the sign, rather than to the right. See "Operators With and Without Limits."
<SUM_NOLIMITS>	Specifies summation that places superscripts and subscripts to the right of the sign, rather than above and below. See "Operators With and Without Limits."
<SUPERSCRIPT> (*expression*)	Exponentiation, as in a^b. See "Operators With and Without Limits."
<TEXT>	Specifies text in a math expression.
<TIMES>	Multiplication, as in $a * b$.
<TO>	Indicates progression, as in $1 \rightarrow 10$.
<VARIABLE>	Specifies a variable name that contains nonalphanumeric characters.
<VDOTS>	Specifies vertical dots, as in $\vdots$.
<VECTOR> (*var*)	A vector, as in $\vec{x}$.
<X_TIMES>	Multiplication, as in $a \times b$.

Operators With and Without Limits

When you specify <SUBSCRIPT> and <SUPERSCRIPT> tags, the superscription or subscription applies to the immediately preceding variable or expression. These tags can be specified in any order, as shown in the following code fragment:

```
<MATH>(a<subscript>(2)<superscript>(n-1))
```

This code fragment may produce the following output: a_2^{n-1}

In extended mathematical expressions, the <SUPERSCRIPT> and <SUBSCRIPT> tags produce differing results in conjunction with the <INTEGRAL> , <PROD> , and <SUM> functions and their complementary tags <INTEGRAL_LIMITS> and <INTEGRAL_NOLIMITS> , <PROD_LIMITS> and <PROD_NOLIMITS> , and <SUM_LIMITS> and <SUM_NOLIMITS> .

The <INTEGRAL> , <PROD> , and <SUM> tags place their subscripts and superscripts depending on whether the tags are used in math text mode (used as an argument to the <MATH> tag) or in math display mode (used between the <MATH> (DISPLAY) and <ENDMATH> tags).

In math text mode, the <INTEGRAL> , <PROD> and <SUM> tags all place the superscript and subscript to the right. These tags are coded as follows:

```
<MATH>(<integral><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>   )
<MATH>(    <prod><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>   )
<math>(     <sum><subscript>(n=1)<superscript>( <pi><over>(2) )        )
```

This code fragment may have the following output:

$\int_{n=1}^{\frac{\pi}{2}} \ \prod_{n=1}^{\frac{\pi}{2}} \ \sum_{n=1}^{\frac{\pi}{2}}$

In math display mode, the <PROD> and <SUM> tags place the superscript and subscript above and below their respective signs; however, the <INTEGRAL> tag places superscripts and subscripts to the right. These tags are coded as follows:

```
<MATH>(display)
<integral><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
    <prod><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
     <sum><subscript>(n=1)<superscript>( <pi><over>(2) )
<ENDMATH>
```

This code fragment may have the following output:

$$\int_{n=1}^{\frac{\pi}{2}} \prod_{n=1}^{\frac{\pi}{2}} \sum_{n=1}^{\frac{\pi}{2}}$$

The <INTEGRAL_NOLIMITS> , <PROD_NOLIMITS> , and <SUM_NOLIMITS> tags always place the subscripts and superscripts to the right of the signs, regardless of the math mode. These tags are coded as follows:

```
<MATH>(display)
<integral_nolimits><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
    <prod_nolimits><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
     <sum_nolimits><subscript>(n=1)<superscript>( <pi><over>(2) )
<ENDMATH>
```

This code fragment may have the following output:

$\int_{n=1}^{\frac{\pi}{2}} \prod_{n=1}^{\frac{\pi}{2}} \sum_{n=1}^{\frac{\pi}{2}}$

The <INTEGRAL_LIMITS> , <PROD_LIMITS> , and <SUM_LIMITS> tags always place the subscripts and superscripts above and below the signs, regardless of the math mode. These tags are coded as follows:

```
<MATH>(display)
<integral_limits><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
    <prod_limits><subscript>(n=1)<superscript>( <pi><over>(2) ) <sp>
     <sum_limits><subscript>(n=1)<superscript>( <pi><over>(2) )
<ENDMATH>
```

This code fragment may have the following output:

$$\int\limits_{n=1}^{\frac{\pi}{2}} \prod\limits_{n=1}^{\frac{\pi}{2}} \sum\limits_{n=1}^{\frac{\pi}{2}}$$

Mathematical Functions

In addition to the operations and special functions listed in Table 9–3, you can specify mathematical functions using any of the tags listed in Table 9–4. These tags all let you specify the tag with or without an argument. If you specify an argument, it is placed in parentheses following the function name. For example:

```
<MATH>(<SIN>(d))
```

This produces: $\sin(d)$.

Table 9–4 Tags for Mathematical Functions

Tag	Function
<ARCCOS>	arccos
<ARCSIN>	arcsin
<ARCTAN>	arctan
<ARG>	arg
<COS>	cos
<COSH>	cosh
<COT>	cot
<COTH>	coth
<CSC>	csc
<DEG>	deg
<DET>	det
<DIM>	dim
<EXP>	exp

Table 9–4 (Cont.) Tags for Mathematical Functions

Tag	Function
<GCD>	gcd
<HOM>	hom
<INF>	inf
<KER>	ker
<LG>	lg
<LIM>	lim
<LIMINF>	lim inf
<LIMSUP>	lim sup
<LN>	ln
<LOG>	log
<MAX>	max
<MIN>	min
<MOD>	mod
<PMOD>	(mod n)
<PR>	Pr
<SEC>	sec
<SIN>	sin
<SINH>	sinh
<SUP>	sup
<TAN>	tan
<TANH>	tanh

For example, the following:

```
<LIST>(UNNUMBERED)

<le><math>(
<sin>2<math_char>(theta)
     <equals>2<sin><math_char>(theta)<cos><math_char>(theta)
)

<le><math>(
O(n <log>n <log><log>n)
)

<le><math>(
<pr>(X >x)= <exp>(-x/<math_char>(mu))
)

<le><math>(
  <max><subscript>(1<math_char>(geq)n<math_char>(geq)m)
  <log><subscript>(2)P<subscript>(n)
)

<le><math>(
<lim><subscript>(x<to>0)<group>(
          <sin>x<over>(x))
<equals>1
)
<ENDLIST>
```

Produce:

- $\sin 2\theta = 2\sin\theta\cos\theta$
- $O(n \log n \log\log n)$
- $\Pr(X > x) = \exp(-x/\mu)$
- $\max_{1\geq n\geq m} \log_2 P_n$
- $\lim_{x\to 0} \frac{\sin x}{x} = 1$

Delimited Expressions

To produce delimited expressions in mathematics, you must use one of the following pairs of tags:

- <LBAR> and <RBAR> —for opening and closing vertical bars
- <LBRACE> and <RBRACE> —for opening and closing curly braces
- <LBRACKET> and <RBRACKET> —for opening and closing square brackets
- <LCEIL> and <RCEIL> —for opening and closing ceiling delimiters
- <LFLOOR> and <RFLOOR> —for opening and closing floor delimiters
- <LPAREN> and <RPAREN> —for opening and closing parentheses.

The text formatter automatically assumes that text within these pairs is to be grouped, and it sizes the delimiters automatically.

For example:

```
<MATH>(display)
<group>(
   C<subscript>(dg)
       ) =

  <fraction>(<math_char>(partial)Q<subscript>(d)\
                          <math_char>(partial)V<subscript>(g))

 =
  -C<subscript>(oxt)
    [0.5
          +
    <lparen>f<subscript>(0) DVG  -
         <group>(2f<subscript>(0) V<subscript>(com)<over>(f<subscript>(1))
         )
    <rparen>
    <group>(1 <over>(f<subscript>(1)<superscript>(2)))
]
<ENDMATH>
```

This produces:

$$C_{dg} = \frac{\partial Q_d}{\partial V_g} = -C_{oxt}\left[0.5 + \left(f_0 DVG - \frac{2f_0 V_{com}}{f_1}\right)\frac{1}{f_1^2}\right]$$

Matrices and Cases

You can construct matrices and case constructs using the tags provided with <MATRIX> and <CASES> .

The <MATRIX> tag has the format:

$$\text{<MATRIX>} \left(\begin{bmatrix} \text{BRACES} \\ \text{BRACKETS} \\ \text{VERTICAL_RULE} \end{bmatrix} \right)$$

Where the keywords BRACES, BRACKETS, and VERTICAL_RULE override the default matrix delimiter, parentheses.

When you construct a matrix, each row in the matrix must be specified using the <MATRIX_ROW> tag. You can specify a maximum of nine columns for the row. The matrix must be terminated with the <ENDMATRIX> tag. For example:

```
<matrix>(brackets)
 <matrix_row>(A)
 <matrix_row>(B)
 <matrix_row>(C)
 <matrix_row>(D)
<endmatrix>
```

This simple, one-column matrix may produce the following output:

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$$

A more complex example shows how to code a multi-column matrix:

```
<math>(display)
<det><matrix>(vertical_rule)
  <matrix_row>(c<subscript>(0)\c<subscript>(1)\
              c<subscript>(2)\<dots>\c<subscript>(n))
  <matrix_row>(c<subscript>(1)\c<subscript>(2)\
              c<subscript>(3)\<dots>\c<subscript>(n+1))
  <matrix_row>(c<subscript>(2)\c<subscript>(3)\
              c<subscript>(4)\<dots>\c<subscript>(n+2))
  <matrix_row>(<vdots>\<vdots>\<vdots>\<vdots>)
  <matrix_row>(c<subscript>(n)\c<subscript>(n+1)\
              c<subscript>(n+2)\<dots>\c<subscript>(2n))
  <endmatrix> > 0.
<endmath>
```

Would produce:

$$\det \begin{vmatrix} c_0 & c_1 & c_2 & \cdots & c_n \\ c_1 & c_2 & c_3 & \cdots & c_{n+1} \\ c_2 & c_3 & c_4 & \cdots & c_{n+2} \\ \vdots & \vdots & \vdots & \vdots & \\ c_n & c_{n+1} & c_{n+2} & \cdots & c_{2n} \end{vmatrix} > 0.$$

<CASES> is similar to <MATRIX> , but it produces only a large left-hand brace; there is no closing delimiter. It is specified as:

```
<MATH>(display)
<cases>
<case_row>(1/3\if\0> x\;)
<case_row>(2/3\if\3< x\;)
<case_row>(0\elsewhere.)
<endcases>
<ENDMATH>
```

This might produce:

$$\begin{cases} 1/3 & \text{if } 0 > x; \\ 2/3 & \text{if } 3 < x; \\ 0 & \text{elsewhere.} \end{cases}$$

EXAMPLES

1

```
The circumference of a circle is calculated using the
formula <MATH>(c<equals><pi>r<superscript>(2)).
```

This example illustrates a simple mathematical expression used in the text of a sentence. This example produces: " The circumference of a circle is calculated using the formula $c = \pi r^2$."

2

```
<MATH>(DISPLAY)
vsize <equals> psize <minus>(<minus>topglue) <minus> topdepth <minus> footerglue
<ENDMATH>
```

This example produces:

$$vsize = psize - (-topglue) - topdepth - footerglue$$

Note that the parenthetical expression following the first <MINUS> tag must have a space in front of it; otherwise the expression will be interpreted as an argument to <MINUS> .

3

```
<LIST>(NUMBERED)
<le><MATH>(1<over>(2))
<le><MATH>(n+1<over>(3))
<le><MATH>(<choose>(N+1\3))
<le><MATH>(<sum><subscript>(n=1)<superscript>(3)Z<subscript>(n)<superscript>(2))
<ENDLIST>
```

This list of simple inline expressions produces:

1. $\frac{1}{2}$
2. $\frac{n+1}{3}$
3. $\binom{N+1}{3}$
4. $\sum_{n=1}^{3} Z_n^2$

4
```
<MATH>(DISPLAY)
<GROUP>(
a<over>(x<plus>y<superscript>(3))
)
   <EQUALS> <SQRT>(<times><pi>)
<ENDMATH>
```

This example illustrates how to use the <GROUP> tag to indicate the order of operation. Its output is:

$$\frac{a}{x+y^3} = \sqrt{*\pi}$$

Note what the output would be if the <GROUP> tag is not present:

$$\frac{a}{x+y^3 = \sqrt{*\pi}}$$

5
```
<MATH>(total = A + B - C * D / E)
```

In this example, the characters representing the mathematical operations are used directly. The output is: $total = A + B - C * D/E$. Note that this is equivalent to:

```
<MATH>(total <EQUALS> A <PLUS> B <MINUS> C <TIMES> D <DIVIDED_BY> E)
```

6
```
<MATH>(--)This begins a comment line.
```

This example illustrates how to use the <MATH> tag to generate true minus signs in your SDML files. This example results in:

−−This begins a comment line.

7
```
<MATH>(DISPLAY\widget_equation)
widgets = crickets + bats
<ENDMATH>
<p>As <REFERENCE>(widget_equation) shows, the relationship between
bats and widgets must include crickets.
```

This example shows how you may use the <REFERENCE> tag to refer to an equation. This example may produce the following output:

$$widgets = crickets + bats \qquad (9\text{–}1)$$

As (9–1) shows, the relationship between bats and widgets must include crickets.

<MATH_CHAR>

Creates a special mathematical symbol.

FORMAT

<MATH_CHAR> *(keyword)*

ARGUMENTS

keyword
A keyword indicating the special symbol you want to access. The keywords, and the symbols they produce, are listed in Table 9–5.

related tags

- <MATH> —provides tags for operations in constructing mathematical expressions.
- <MCS> —provides access to special characters available in the DEC Multinational Character Set.
- <SPECIAL_CHAR>

restrictions

- The <MATH_CHAR> tag is invalid in the monospaced example tags.
- The characters produced using <MATH_CHAR> are sized only for normal text sizes and therefore will not produce good visual results in header levels, text in the OVERHEADS doctype, and so on.

required terminator

None.

DESCRIPTION

Table 9–5 summarizes the keywords and special symbols you can access with the <MATH_CHAR> tag.

Table 9–5 <MATH_CHAR> Symbols

Keyword	Symbol
The Greek Letters	
ALPHA	α
BETA	β
GAMMA	γ
DELTA	δ
EPSILON	ϵ
VAREPSILON	ε
ZETA	ζ
ETA	η
THETA	θ

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
The Greek Letters	
VARTHETA	ϑ
IOTA	ι
KAPPA	κ
LAMBDA	λ
MU	μ
NU	ν
XI	ξ
OMICRON	o
PI	π
VARPI	ϖ
RHO	ρ
VARRHO	ϱ
SIGMA	σ
VARSIGMA	ς
TAU	τ
UPSILON	υ
PHI	ϕ
VARPHI	φ
CHI	χ
PSI	ψ
OMEGA	ω
The Uppercase Greek Letters	
UPPERCASE_ALPHA	A
UPPERCASE_BETA	B
UPPERCASE_GAMMA	Γ
UPPERCASE_DELTA	Δ
UPPERCASE_EPSILON	E
UPPERCASE_ZETA	Z
UPPERCASE_ETA	H
UPPERCASE_THETA	Θ
UPPERCASE_IOTA	I
UPPERCASE_KAPPA	K
UPPERCASE_LAMBDA	Λ
UPPERCASE_MU	M
UPPERCASE_NU	N
UPPERCASE_XI	Ξ

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
The Uppercase Greek Letters	
UPPERCASE_OMICRON	O
UPPERCASE_PI	Π
UPPERCASE_RHO	P
UPPERCASE_SIGMA	Σ
UPPERCASE_TAU	T
UPPERCASE_UPSILON	Υ
UPPERCASE_PHI	Φ
UPPERCASE_CHI	X
UPPERCASE_PSI	Ψ
UPPERCASE_OMEGA	Ω
The Ordinal Operators	
ALEPH	$\aleph$
ANGLE	$\angle$
BACKSLASH	$\backslash$
BOT	$\bot$
CLUBSUIT	$\clubsuit$
DIAMONDSUIT	$\diamondsuit$
DOUBLE_VERT	$\|$
ELL	ℓ
EMPTYSET	$\emptyset$
EXISTS	$\exists$
FLAT	$\flat$
FORALL	$\forall$
HBAR	$\hbar$
HEARTSUIT	$\heartsuit$
IMATH	$\imath$
IM	$\Im$
INFTY	∞
JMATH	$\jmath$
NABLA	∇
NATURAL	$\natural$
NEG	$\neg$
PARTIAL	∂
PRIME	$\prime$
RE	$\Re$
SHARP	$\sharp$

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
The Ordinal Operators	
SPADESUIT	♠
SURD	√
TOP	⊤
TRIANGLE	△
WP	℘
Binary Operators	
AMALG	⨿
AST	∗
BIGCIRC	◯
BIGTRIANGLEDOWN	▽
BIGTRIANGLEUP	△
BULLET	•
CAP	∩
CDOT	·
CIRC	∘
CUP	∪
DAGGER	†
DDAGGER	‡
DIAMOND	⋄
DIV	÷
MP	∓
ODOT	⊙
OMINUS	⊖
OPLUS	⊕
OSLASH	⊘
OTIMES	⊗

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
Binary Operators	
PM	$\pm$
SETMINUS	$\setminus$
SQCAP	$\sqcap$
SQCUP	$\sqcup$
STAR	$\star$
TIMES	$\times$
TRIANGLELEFT	$\triangleleft$
TRIANGLERIGHT	$\triangleright$
UPLUS	$\uplus$
VEE	$\vee$
WEDGE	$\wedge$
WR	$\wr$

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
Relational Operators	
APPROX	$\approx$
ASYMP	$\asymp$
BOWTIE	$\bowtie$
CONG	$\cong$
DASHV	$\dashv$
DOTEQ	$\doteq$
EQUIV	$\equiv$
FROWN	$\frown$
GEQ	$\geq$
GG	$\gg$
IN	$\in$
LL	$\ll$
LEQ	$\leq$
MID	$\mid$
MODELS	$\models$
NI	$\ni$
NOTSUBSETEQ	$\not\subseteq$
NOT_APPROX	$\not\approx$
NOT_ASYMP	$\not\asymp$
NOT_CONG	$\not\cong$
NOT_EQUIV	$\not\equiv$
NOT_EQ	$\neq$
NOT_GEQ	$\not\geq$

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
Relational Operators	
NOT_GT	$\not>$
NOT_LEQ	$\not\leq$
NOT_LT	$\not<$
NOT_PRECEQ	$\not\preceq$
NOT_PREC	$\not\prec$
NOT_SIMEQ	$\not\simeq$
NOT_SIM	$\not\sim$
NOT_SQSUBSETEQ	$\not\sqsubseteq$
NOT_SQSUPSETEQ	$\not\sqsupseteq$
NOT_SUBSET	$\not\subset$
NOT_SUCCEQ	$\not\succeq$
NOT_SUCC	$\not\succ$
NOT_SUPSETEQ	$\not\supseteq$
NOT_SUPSET	$\not\supset$
PARALLEL	$\parallel$
PERP	$\perp$
PRECEQ	$\preceq$
PREC	$\prec$
PROPTO	$\propto$
SIMEQ	$\simeq$
SIM	$\sim$
SMILE	$\smile$
SQSUBSETEQ	$\sqsubseteq$
SQSUPSETEQ	$\sqsupseteq$
SUBSETEQ	$\subseteq$
SUBSET	$\subset$
SUCCEQ	$\succeq$
SUCC	$\succ$
SUPSETEQ	$\supseteq$
SUPSET	$\supset$
VDASH	$\vdash$

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
Arrows	
DOUBLE_DOWNARROW	⇓
DOUBLE_LEFTARROW	⇐
DOUBLE_LEFTRIGHTARROW	⇔
DOUBLE_LONGLEFTARROW	⟸
DOUBLE_LONGLEFTRIGHTARROW	⟺
DOUBLE_LONGRIGHTARROW	⟹
DOUBLE_RIGHTARROW	⇒
DOUBLE_UPARROW	⇑
DOUBLE_UPDOWNARROW	⇕
DOWNARROW	↓
HOOKLEFTARROW	↩
HOOKRIGHTARROW	↪
LEFTARROW	←
LEFTHARPOONDOWN	↽
LEFTHARPOONUP	↼
LEFTRIGHTARROW	↔
LONGLEFTARROW	⟵
LONGLEFTRIGHTARROW	⟷
LONGMAPSTO	⟼
LONGRIGHTARROW	⟶
MAPSTO	↦
NEARROW	↗
NWARROW	↖
RIGHTARROW	→
RIGHTHARPOONDOWN	⇁
RIGHTHARPOONUP	⇀
RIGHTLEFTHARPOONS	⇌
SEARROW	↘
SWARROW	↙
UPARROW	↑
UPDOWNARROW	↕

Table 9–5 (Cont.) <MATH_CHAR> Symbols

Keyword	Symbol
Delimiters	
LANGLE	⟨
RANGLE	⟩
LBRACE	{
RBRACE	}
LBRACK	[
RBRACK	]
LCEIL	⌈
RCEIL	⌉
LFLOOR	⌊
RFLOOR	⌋

<MCS>

Labels a character in the DEC Multinational Character Set.

FORMAT

<MCS> *(character)*

ARGUMENTS

character

Specifies the character referred to. The character may be any one of the following:

Argument	Character	Decimal Value
spanish_inverted_exclamation	¡	161
cents	¢	162
british_pound	£	163
—	[reserved]	164
japanese_yen	¥	165
—	[reserved]	166
section_sign	§	167
general_currency	¤	168
copyright	©	169
feminine_ordinal	ª	170
double_open_angle_brackets	«	171
—	[reserved]	172
—	[reserved]	173
—	[reserved]	174
—	[reserved]	175
degree	°	176
plus_or_minus	±	177
superscript2	²	178
superscript3	³	179
—	[reserved]	180
micro	µ	181
pilcrow	¶	182
raised_period	·	183
—	[reserved]	184
superscript1	¹	185
masculine_ordinal	º	186
double_close_angle_brackets	»	187
one_fourth	¼	188

Argument	Character	Decimal Value
one_half	½	189
—	[reserved]	190
spanish_inverted_question	¿	191
cap_a_grave	À	192
cap_a_acute	Á	193
cap_a_circumflex	Â	194
cap_a_tilde	Ã	195
cap_a_umlaut	Ä	196
cap_a_ring	Å	197
cap_ae	Æ	198
cap_c_cedilla	Ç	199
cap_e_grave	È	200
cap_e_acute	É	201
cap_e_circumflex	Ê	202
cap_e_umlaut	Ë	203
cap_i_grave	Ì	204
cap_i_acute	Í	205
cap_i_circumflex	Î	206
cap_i_umlaut	Ï	207
—	[reserved]	208
cap_n_tilde	Ñ	209
cap_o_grave	Ò	210
cap_o_acute	Ó	211
cap_o_circumflex	Ô	212
cap_o_tilde	Õ	213
cap_o_umlaut	Ö	214
cap_oe	Œ	215
cap_o_slash	Ø	216
cap_u_grave	Ù	217
cap_u_acute	Ú	218
cap_u_circumflex	Û	219
cap_u_umlaut	Ü	220
cap_y_umlaut	Ÿ	221
—	[reserved]	222
german_ss	ß	223
small_a_grave	à	224
small_a_acute	á	225
small_a_circumflex	â	226
small_a_tilde	ã	227

Argument	Character	Decimal Value
small_a_umlaut	ä	228
small_a_ring	å	229
small_ae	æ	230
small_c_cedilla	ç	231
small_e_grave	è	232
small_e_acute	é	233
small_e_circumflex	ê	234
small_e_umlaut	ë	235
small_i_grave	ì	236
small_i_acute	í	237
small_i_circumflex	î	238
small_i_umlaut	ï	239
—	[reserved]	240
small_n_tilde	ñ	241
small_o_grave	ò	242
small_o_acute	ó	243
small_o_circumflex	ô	244
small_o_tilde	õ	245
small_o_umlaut	ö	246
small_oe	œ	247
small_o_slash	ø	248
small_u_grave	ù	249
small_u_acute	ú	250
small_u_circumflex	û	251
small_u_umlaut	ü	252
small_y_umlaut	ÿ	253
—	[reserved]	254

related tags

- <MATH_CHAR>
- <SPECIAL_CHAR>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <MCS> tag labels a character in the DEC Multinational Character Set. Note that this tag is required only when the input device (terminal) you are using does not accept or display these special characters.

EXAMPLE

```
<P>The DEC Multinational Character Set includes the currency sign
for the Japanese yen (<MCS>(japanese_yen)).
```

This example may produce the following output:

The DEC Multinational Character Set includes the currency sign for the Japanese yen (¥).

<NESTED_TABLE_BREAK>

Marks a place that a nested table may be broken across pages.

FORMAT

<NESTED_TABLE_BREAK>

ARGUMENTS

None.

related tags

- <TABLE>
- <TABLE_FILE>
- <TABLE_SPACE>
- <REFERENCE>

restrictions

The <NESTED_TABLE_BREAK> tag must be used within a nested table, that is, a table begun within a table row. Also, breakpoints for long tables nested inside multipage tables have the following limitations:

- The first level table must not use the KEEP attribute.
- The first level table must be set up with only two or three columns.

required terminator

None.

DESCRIPTION

The <NESTED_TABLE_BREAK> tag marks a place that a nested table may be broken across pages. A nested table is a table that is coded within another table. You create a nested table by placing the appropriate table tags within an argument to the <TABLE_ROW> tag of the outer table. See the Example section in this tag description for an example of this coding.

If you place a <NESTED_TABLE_BREAK> tag in a nested table that contains headings, the headings are not repeated on subsequent pages. Also, a rule may be output at the point that the tag is included, thereby appearing to end the nested table which continues on the next page.

EXAMPLE

```
<TABLE>(Table Caption\tab_log_name)
<TABLE_SETUP>(2\10)
<TABLE_HEADS>(First Head\Second Head)
<TABLE_ROW>(Item Here\Definition for Item here.)

<TABLE_ROW>(Item Here\Definition that tells about nested table.
        <COMMENT>(*nested table begins*)

        <TABLE>
        <TABLE_SETUP>(2\15)
        <TABLE_ROW>(Item\Text of item description here)
        <TABLE_ROW>(Item\Text of item description here)
            .
    .
    .
        <TABLE_ROW>(Item\Text of item description here)
        <TABLE_ROW>(Item\Text of item description here)
        <NESTED_TABLE_BREAK>
        <TABLE_ROW>(Item\Text of item description here)
            .
    .
    .
        <TABLE_ROW>(Item\Text of item description here)
        <NESTED_TABLE_BREAK>
        <TABLE_ROW>(Item\Text of item description here)
            .
    .
    .
        <TABLE_ROW>(Item\Text of item description here)
            .
    .
    .
        <TABLE_ROW>(Item Here\Definition for Item here.)
        <TABLE_ROW>(Item Here\Definition for Item here.)
        <ENDTABLE>

        <COMMENT>(*nested table ends*)
)
<TABLE_ROW>(Item Here\Definition for Item here.)
<TABLE_ROW>(Item Here\Definition for Item here.)
<TABLE_ROW>(Item Here\Definition for Item here.)
<ENDTABLE>
```

This example illustrates the coding of the <NESTED_TABLE_BREAK> tag.

<NEWTERM>

Labels a term first introduced into the text in order to emphasize the term.

FORMAT

<NEWTERM> *(term)*

ARGUMENTS

term
Specifies the word just introduced.

related tags

- <KEYWORD>
- <VARIABLE>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <NEWTERM> tag labels a term first introduced into the text in order to emphasize the term. In output, the term will be italicized.

EXAMPLE

```
<P>To begin a session at the terminal, you must first <NEWTERM>(log in).
Logging in consists of getting the system's attention and identifying yourself as
an authorized user.
```

This example may produce the following output:

To begin a session at the terminal, you must first *log in.* Logging in consists of getting the system's attention and identifying yourself as an authorized user.

<NOTE>

Labels a note, caution, warning, or some other portion of text to which you wish to draw attention.

FORMAT

<NOTE> *[(heading-text)]*
note-text

.
.
.

<ENDNOTE>

ARGUMENTS

heading-text

Specifies text for a heading other than the default heading "Note:".

note-text

Specifies the text of the note. A paragraph is implied here, so no <P> tag is needed.

related tags

None.

restrictions

None.

required terminator

<ENDNOTE>

DESCRIPTION

The <NOTE> tag labels a note, caution, warning, or some other portion of text to which you wish to draw attention. A note will be formatted differently depending on the doctype specified on the DOCUMENT command line.

EXAMPLE

```
<NOTE>(Caution)You should abort the system generation command
procedure only after Phase 1 has completed processing.
<ENDNOTE>
```

This example might produce the following output, depending on the doctype specified:

Caution: **You should abort the system generation command procedure only after Phase 1 has completed processing.**

<OPAREN>

Supplies an unmatched opening parenthesis in an argument to a tag.

FORMAT

<OPAREN>

ARGUMENTS

None.

related tags

- The following tags label other characters that must be tagged when they occur in an argument to a global tag:

  <AMPERSAND>
  <BACKSLASH>
  <CPAREN>
  <VBAR>

restrictions

None.

required terminator

None.

DESCRIPTION

The <OPAREN> tag can be used anywhere to insert an open parenthesis character into text. However, it is only beneficial (in terms of keystrokes and control of the output) as an unmatched opening parenthesis in an argument passed to a tag.

An unmatched parenthesis in an argument can cause errors when processed because the parentheses are used to determine the beginning and ending of an argument list. The <OPAREN> tag inserts the opened parenthesis character but is not evaluated as an opened parenthesis.

EXAMPLE

```
<SUBHEAD1>(Using an Opened Parenthesis <PARENDCHAR>(<OPAREN>) in an Argument to a Tag)
```

This example may produce the following output:

Using an Opened Parenthesis (() in an Argument to a Tag

<ORDER_NUMBER>

Labels the order number or part number that may appear on the title page of a book.

FORMAT

<ORDER_NUMBER> *(number)*

ARGUMENTS

number
Specifies the order number for the book.

related tags

None.

restrictions

The <ORDER_NUMBER> tag is valid only within a title page.

required terminator

None.

DESCRIPTION

The <ORDER_NUMBER> tag labels the order number or part number that may appear on the title page of a book.

EXAMPLE

See the example in the <FRONT_MATTER> discussion.

<P>

Marks the beginning of a new paragraph.

FORMAT

<P>

ARGUMENTS

None.

related tags

- <CP>
- <LINE>
- <CENTER_LINE>
- <RIGHT_LINE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <P> tag marks a new paragraph of text. An internal counter keeps track of the context in which the new paragraph begins, so that you can freely use <P> tags inside lists and in other contexts where you want to start a new paragraph, while maintaining the same logical level of discussion.

A <P> tag is expected after every heading. If you do not immediately start a new paragraph after a heading, you must label the beginning of another text element, such as a list.

EXAMPLES

1

```
<P>
Here is a sentence or two in a paragraph. The following paragraph
will show the <TAG>(P) tag at work separating paragraphs.
<P>Here is the second paragraph so that you can observe the relationship
it has to the first.
```

This example may produce the following output, depending on the doctype specified:

Here is a sentence or two in a paragraph. The following paragraph will show the <P> tag at work separating paragraphs.

Here is the second paragraph so that you can observe the relationship it has to the first.

2
```
<list>(unnumbered)
              <le>Oranges
              <le>Apples
            <p>Note that there are several types of apples<hellipsis>
              <le>Bananas
<endlist>
```

This example shows how a paragraph can be used within a list. This example may produce the following output:

- Oranges
- Apples

  Note that there are several types of apples . . .
- Bananas

<PAGE>

Breaks a page of text, forcing the text that follows the tag to begin on a new page.

FORMAT

<PAGE> *[({ EVEN | ODD })]*

ARGUMENTS

EVEN
ODD

Specifies whether the new page of output should have an even or an odd page number.

Note that if you use one of these arguments, you might have two consecutive pages of output numbered such as 13 or 15. There will be no blank page of output between the pages.

related tags

- <FINAL_CLEANUP> —page_break

restrictions

The arguments EVEN and ODD are invalid in tables.

Invalid in math.

required terminator

None.

DESCRIPTION

The <PAGE> tag breaks a page of text, forcing the text that follows the tag to begin on a new page.

Note: Use this tag only in special cases, where a page must be broken at that point. This tag is not for general use, such as for overall pagination; this can jeopardize the output format and device independence of your generically coded source file.

EXAMPLE

```
<HEAD1>(Survey Results)
   .
   .
   .
<PAGE>
<HEAD2>(Brand X)
   .
   .
   .
<PAGE>
<HEAD2>(Brand Z)
   .
   .
   .
```

This example shows the coding of a short document in which the information might be easier to index when specific headings start on new pages.

<PARENDCHAR>

Labels a character that will appear alone within parentheses to achieve better spacing.

FORMAT

<PARENDCHAR> *(char)*

ARGUMENTS

char

Specifies the character within parentheses. Specify only the character. The parentheses will be added during processing.

related tags

None.

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

Frequently writers use the name for a special character followed by that character in parentheses. For instance, in discussing wildcard characters it is natural to say, "The percent sign (%) is the wildcard for a single character."

A problem in using proportionally spaced fonts is that single characters surrounded by parentheses can look quite crowded, like this: "The percent sign (%) is the wildcard for a single character." (Compare the percent sign in parentheses in this paragraph to the percent sign in parentheses in the previous paragraph; you can see the slight crowding at the top left corner of the percent sign.)

Typographers put a small amount of space, called a "thin space," between the parentheses and the single character to achieve a balance. When you use the <PARENDCHAR> tag, that thin space is added for you.

You should use the <PARENDCHAR> tag to label characters that are small or that are more vertical than horizontal in shape.

When using the <PARENDCHAR> tag, do not enter the parentheses that will surround the character. The parentheses will be added during processing.

EXAMPLE

```
<P>The tilde <PARENDCHAR>(~) is used to prevent the word
following it from becoming a main entry in the index.
```

This example may produce the following output:

The tilde (~) is used to prevent the word following it from becoming a main entry in the index.

Without the <PARENDCHAR> tag, your output would look like this:

The tilde (~) is used to prevent the word following it from becoming a main entry in the index.

<PART>

Labels the start of a major division within a document, and starts it on a new page.

FORMAT

<PART> *[(zone-title \ symbol-name)]*

ARGUMENTS

zone-title

Specifies the title for this part of your document.

symbol-name

Specifies the term that you assign to this part and then use to reference the part throughout your document. The symbol-name argument is required only if the part will be included in a bookbuild.

Symbol-names must not exceed 31 characters, and must only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

restrictions

None.

required terminator

None.

DESCRIPTION

A document may be logically divided into units called "Parts," which may consist of the following:

- A set of chapters or appendixes whose contents are logically related.
- A collection of units of logically related information, for example, reference items in the SOFTWARE doctype.

The <PART> tag begins a new part and optionally assigns a title to it. Like chapters, parts are automatically numbered. If you specify a symbol-name argument, a symbol table entry is created for the part.

The <PART_PAGE> tag creates a divider page for a part and optionally provides an abstract describing information in the part.

By default, the part page is assigned the next odd page number in the current numbering sequence in a document that does not have chapters. If RENUMBER is specified in the <ENDPART_PAGE> tag, the part page is assigned the page number of 1, and the next printed page of output is numbered page 3.

EXAMPLES

1

```
<part>(Reference Information\ref_part)
<part_page>
<title>(<reference>(ref_part)\<reference>(ref_part\text))
<abstract>
This part contains reference chapters. <reference>(intro_part)
contains tutorial information.
<endabstract>
<endpart_page>
```

This example shows how to code the tags <PART> and <PART_PAGE>, assigning a zone-title and a symbol-name to a part.

2

```
<part>(Introduction\intro_part)
<part_page>
<title>(<reference>(intro_part)\<reference>(intro_part\text))
<abstract>
This part contains introductory chapters. <reference>(ref_part)
contains more detailed reference information.
<endabstract>
<endpart_page>
```

This example results in the following if this is the first part specified in the document.

- A symbol table entry for the symbol-name *intro_part,* whose value is 1 and whose text is "Introduction."
- The <PART_PAGE> tag creates a new page of output. The page is assigned the next odd page number in the numbering sequence.
- The title: "PART I Introduction" will be printed at the top of the page. The use of uppercase roman numerals to display the part number is a document-type-specific design attribute.
- The abstract text is printed.

<PART_PAGE>

Begins a divider page for a new part of a document.

FORMAT

<PART_PAGE>

ARGUMENTS

None.

related tags

- <TITLE>
- <RUNNING_TITLE>
- <ABSTRACT>

restrictions

None.

required terminator

<ENDPART_PAGE>

DESCRIPTION

The <PART> tag divides a document into one of its major constituent parts. The <PART_PAGE> tag (together with <ENDPART_PAGE>) simply inserts an extra page in between major sections.

By labeling a part page, you also define three related tags: <TITLE> , <ABSTRACT> , and <RUNNING_TITLE> . You can use these related tags to put either a title or an abstract on the page (or both), and you can determine the running title for the text that follows the part page as well.

EXAMPLE

```
<PART_PAGE>
<TITLE>(Part II)
<ABSTRACT>
Part II provides reference information on each global tag in SDML.
<ENDABSTRACT>
<RUNNING_TITLE>(Descriptions of Global Tags)
<ENDPART_PAGE>
```

This example illustrates the use of both the <PART_PAGE> and <ENDPART_PAGE> tags, and the related tags that can be used to put a title on the page, print a brief abstract, and set the running title for the text that follows.

<PREFACE>

Labels the beginning of a preface.

FORMAT

<PREFACE> *[(page-number)]*

ARGUMENTS

page-number

Specifies the page number on which you wish the preface to begin. You must specify the number with an Arabic numeral (the number in the formatted result will be a lowercase Roman numeral). If you do not specify a number, the default page on which to begin the preface is Roman numeral five.

The page-number can be either a positive or negative value.

related tags

None.

restrictions

This tag must be used in the context of a <FRONT_MATTER> tag.

required terminator

<ENDPREFACE>

DESCRIPTION

If the preface of a document is placed after the table of contents, you must specify the starting page number for the preface. You cannot know what this number will be until the document is nearly completed and you know what the page count is for the table of contents.

The table of contents file usually begins on page 3, which is normally output as iii. When you have examined the table of contents output, you then can know the table of contents' last page. Specify the next odd number to set the correct page number for the preface.

EXAMPLE

See the example in the discussion of the <FRONT_MATTER> tag.

<PREFACE_SECTION>

Creates a major section in the preface of a book to provide information such as a summary of changes to the book.

FORMAT

<PREFACE_SECTION> *(title)*

ARGUMENTS

title

Specifies the title given to this preface section.

related tags

- <PREFACE>

restrictions

The <PREFACE_SECTION> tag is valid only within the front matter.

required terminator

None.

DESCRIPTION

The <PREFACE-SECTION> tag creates a major section in the preface of a book to provide information such as a summary of changes to the book.

EXAMPLE

See the example in the discussion of the <FRONT_MATTER> tag.

<PRINT_DATE>

Inserts a print date line on the copyright page.

FORMAT

<PRINT_DATE> *(date)*

ARGUMENTS

date

Specifies official printing date information for the book.

related tags

None.

restrictions

The <PRINT_DATE> tag is valid only within a copyright page, within <FRONT_MATTER> and <ENDFRONT_MATTER> .

required terminator

None.

DESCRIPTION

The <PRINT_DATE> inserts a print date line on the copyright page.

EXAMPLE

See the example in the <FRONT_MATTER> tag.

<PROFILE>

Indicates that the source file is a profile and that a book build is to be performed.

FORMAT

<PROFILE>

ARGUMENTS

None.

related tags

- <ELEMENT>
- <INCLUDES_FILE>

restrictions

None.

required terminator

<ENDPROFILE>

DESCRIPTION

A profile of a book is required in order to build (process) a book. A book's profile begins with a <PROFILE> tag and ends with the <ENDPROFILE> tag. Between the <PROFILE> and <ENDPROFILE> tags, each element of the book is introduced with an <ELEMENT> tag.

Only those files listed with <ELEMENT> tags are included in the book during the bookbuild. They should be listed in the profile in the order in which they are presented in the book.

Within a profile file, never include a tag that contains text. Also, do not place an <INCLUDE> tag in a profile.

There are several optional tags that can be included between the <PROFILE> and <ENDPROFILE> tags, other than <ELEMENT> tags. These include the following:

- <INDEX_FILE>
- <CONDITION> and <ENDCONDITION>
- <CONTENTS_FILE>
- <COMMENT> and <ENDCOMMENT>
- <INCLUDES_FILE>

See Chapter 4 for more information on bookbuilding.

EXAMPLE

```
<PROFILE>       <COMMENT>(***Profile for How to Use a Computer***)

<CONTENTS_FILE> <COMMENT>(***insert table of contents here***)
<ELEMENT>(Mydisk:[Mydirectory]intro_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]applications_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]tools_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]conclusion_chap.sdml)
<ELEMENT>(Mydisk:[Mydirectory]questions_app.sdml)

<INDEX_FILE> <COMMENT>(***insert index here***)
<ENDPROFILE>
```

This example illustrates the profile of a book that contains a table of contents, four chapters, an appendix and an index. The profile shown in this example must be in a file by itself. The file might be called by a name that indicates it is a profile, such as "COMPUTER_PROFILE.SDML."

Each of the files named in the <ELEMENT> tags should begin with one of the book element tags, such as <CHAPTER> (Introduction\introduction_chap).

<QUOTE>

Labels quoted material in the output.

FORMAT

<QUOTE> *(quote-text)*
or

<QUOTE>
quote-text

.
.
.

<ENDQUOTE>

ARGUMENTS

quote-text
Specifies the quoted text.

related tags

- The following tags are used to label single occurrences of a quotation mark:

  <DOUBLE_QUOTE>
  <SINGLE_QUOTE>

restrictions

None.

required terminator

<ENDQUOTE> —Required if the text is not passed as an argument.

DESCRIPTION

The reason for the <QUOTE> tag lies in a mismatch between conventions used in the typesetting world and conventions used in the computing world. Your terminal keyboard has two possible characters you can use for quotation marks: " (ASCII 34) and ' (ASCII 39). Neither distinguishes between open quotation marks and close quotation marks. You use the same character twice to enclose a word in quotation marks:

"a quoted string"

Typesetters, on the other hand, scrupulously distinguish between open quotation marks and closed quotation marks. The former look like small sixes; the latter, like small nines. Look carefully at a typeset document and you will immediately see the difference between open and close quotation marks.

In order to obtain the proper quotation marks in your typeset output, you must label material inside quotation marks with the <QUOTE> tag. If you use the keyboard quote character instead of the <QUOTE> tag, the open and close quotes in your typeset document will both look like small, double nines. If you use the tag, your output will contain the proper opening and closing quotation marks for the font in which your text is set.

EXAMPLES

1

```
<P>The symbol B will be defined to be the string <QUOTE>(April
showers).
```

This example may produce the following output:

The symbol B will be defined to be the string "April showers."

Note the placement of the period at the end of the sentence. If you want a punctuation mark enclosed outside the quotes, you must not make the punctuation mark part of the argument to the <QUOTE> tag.

2

```
<P>Abraham Lincoln once wrote, <QUOTE> I claim not to have controlled
events, but confess plainly that events have controlled me. <ENDQUOTE>
```

This example may produce the following output:

Abraham Lincoln once wrote, " I claim not to have controlled events, but confess plainly that events have controlled me. "

<REFERENCE>

Makes a reference to a symbol-name in a book element or text element. When processed, <REFERENCE> is replaced with the current value of the symbol-name.

FORMAT

<REFERENCE> *(symbol-name[\ keyword])*

ARGUMENTS

symbol-name

Specifies the name of a symbol assigned in a book element or text element tag (for example, <HEADx> or <TABLE>).

Make sure that the symbol-names, that you define in the book element or text element tags, do not exceed 31 characters, and only contain alphabetic letters, numbers, or underscores in them. Do not begin a symbol-name with an underscore.

keyword

Specifies an optional keyword indicating the type of text replacement to occur in conjunction with the reference:

Keyword	Reference Output
null	Outputs the current value of the symbol and its related symbol type text, for example "Figure 3-2." This is the default.
VALUE	Outputs only the current value of the symbol, for example, the heading level number or the table number, without any related text.
TEXT	Outputs only the text associated with the symbol table entry, that is, the caption (for figures, tables, and examples), or the heading-text (for chapter, section and heading-level tags).
FULL	Outputs all information associated with the symbol, including the value, type, and text.

related tags

None.

restrictions

None.

required terminator

None.

DESCRIPTION

This tag refers to a symbol-name that is specified as an argument in a text or book element. When processed, the <REFERENCE> tag is replaced with the current value of the symbol-name that is stored in the cross-reference file.

A second argument to the tag controls the exact output of the reference. When you do not specify a second argument, the output defaults according to the symbol type. Table 9–6 summarizes the default output for each type of text or book element.

Table 9–6 Element Types and Default Output of Symbol-Names

	Second Argument to <REFERENCE>			
Text Element	**(null)**	**VALUE**	**TEXT**	**FULL**
BOOK TITLE	The title argument specified in <DEFINE_BOOK_NAME> .			
EXAMPLE	Example number	number	Caption argument	Example number, Caption argument
FIGURE	Figure number	number	Caption argument	Figure number, Caption argument
HEADx	Heading number	number	Caption argument	Heading number, Caption argument
TABLE	Table number	number	Caption argument	Table number, Caption argument
TEXT STRING	The text-string argument specified in <DEFINE_SYMBOL> .			
Book Elements	**(null)**	**VALUE**	**TEXT**	**FULL**
APPENDIX	Appendix letter	letter	Title argument	Appendix letter, Title argument
CHAPTER	Chapter number	number	Title argument	Chapter number, Title argument
FRONT MATTER	None	None	None	None
GLOSSARY	None	None	None	None
PART	Part number	number	Title argument	Part number, Title argument

See Chapter 6 for more information on symbol-names and cross-referencing.

EXAMPLES

1

```
<REFERENCE>(lognames_tab) shows ten default system logical names.
<TABLE>(system logical name\lognames_tab)
```

This example shows the use of <REFERENCE> and <TABLE> to reference the symbol-name of a table. As shown, the symbol-name "lognames_tab" was defined in <TABLE> and then referenced with the <REFERENCE> tag. This example has the following output:

Table 1-4 shows ten default system logical names.

2
```
<P>See Chapters <REFERENCE>(lognames_chap\VALUE) and
<REFERENCE>(filespec_chap\VALUE) to see  . . .
```

This example shows the use of the tag <REFERENCE> to make a chapter number reference without outputting any related information, such as the chapter title. This example has the following output:

See Chapters 3 and 4 to see . . .

<REVISION>

Indicates that the document contains either new or modified information and enables the output of the <MARK> tag.

FORMAT

<REVISION> *[(UPDATE [\ update-info])]*

ARGUMENTS

UPDATE

Must be specified if the document is being produced for an update. If you specify UPDATE, the file must contain the <UPDATE_RANGE> and <ENDUPDATE_RANGE> tags to indicate the pages to be processed. If no <UPDATE_RANGE> tags are present, no output will be produced.

update-info

Specifies information that is related to the system version and the date of the update. If specified, this text appears on the bottom of each page of output in the update.

related tags

- <MARK>
- <UPDATE_RANGE>

restrictions

None.

required terminator

None.

DESCRIPTION

The <REVISION> tag can be used in any document type for which you want to indicate to reviewers new or changed information.

This tag enables the <MARK>, <ENDMARK>, <UPDATE_RANGE>, and <ENDUPDATE_RANGE> tags. By default, these tags are defined for all document types to be non-operational; thus, if a file is processed without the <REVISION> tag, they produce no output.

When a file that contains the <REVISION> (UPDATE) tag is processed, the table of contents and index are handled as follows:

- If the <CONTENTS_FILE> or <INDEX_FILE> tag occurs between the bounds of <UPDATE_RANGE> and <ENDUPDATE_RANGE>, and /CONTENTS or /INDEX is specified on the command line, the contents file or index file is included within the pages for the update range.
- If the <CONTENTS_FILE> or <INDEX_FILE> tag occurs in the file but is specified outside the bounds of <UPDATE_RANGE> and <ENDUPDATE_RANGE> tags, and if /CONTENTS or /INDEX is specified on the command line, the table of contents file or index file is included in the output file.

EXAMPLES

1

```
<DOCTYPE>(SOFTWARE)
<REVISION>
   .
   .
   .
<P>
The following characters are legal in MACRO-11 source programs:
<LIST>(UNNUMBERED)
<LE>The letters A through Z. Both upper- and lowercase letters are
acceptable, although, upon input, lowercase  letters are converted to
uppercase.
<MARK>
<LE>Characters in the DEC multinational character set (MCS). A chart
showing the MCS is located in <REFERENCE>(mcs_app),
with a list of directives that support the MCS.
<LE>The digits 0 through 9.
<ENDMARK>
<LE>The characters period <PARENDCHAR>(.) and dollar sign
<PARENDCHAR>($). These characters  are reserved for use as Digital
Equipment Corporation system program symbols.
<ENDLIST>
   .
   .
   .
```

This example shows the use of the <REVISION> tag in a document that is extensively revised. The formatted output may be as follows:

The following characters are legal in MACRO–11 source programs:

- The letters A through Z. Both upper- and lowercase letters are acceptable, although, upon input, lowercase letters are converted to uppercase.
- Characters in the DEC multinational character set (MCS). A chart showing the MCS is located in Appendix A, with a list of directives that support the MCS.
- The digits 0 through 9.
- The characters period (.) and dollar sign ($). These characters are reserved for use as Digital Equipment Corporation system program symbols.

2

```
<REVISION>(UPDATE\July 1986)
   .
   .
   .
<UPDATE_RANGE>(3\10)
```

This example shows the use of the <REVISION> tag in a manual that was updated in July of 1986. See the <UPDATE_RANGE> tag for more information on identifying a section of updated material.

<REVISION_INFO>

Labels a section on a title page that provides information on what previous books have been superseded by the current one.

FORMAT

<REVISION_INFO> *([title-text \ ]information)*

ARGUMENTS

title-text

Provides heading information. If this argument is not specified, the default text "Revision/Update Information" is supplied.

information

Provides revision and update information.

related tags

None.

restrictions

The <REVISION_INFO> tag is valid only within a title page.

required terminator

None.

DESCRIPTION

The <REVISION_INFO> tag labels a section on a title page that provides information on what previous books have been superseded by the current one.

EXAMPLE

See the example in the discussion of the <FRONT_MATTER> tag.

<RIGHT_LINE>

Specifies a line of text that is to be right-justified in the current text margin.

FORMAT

<RIGHT_LINE> *(text [\ { BIGSKIP | SMALLSKIP }])*

ARGUMENTS

text

Specifies a line of text to be set on the right-hand side of the page.

BIGSKIP
SMALLSKIP

Specifies that a set amount of vertical space is to precede the element identified as a line or block of text. The actual amount of space inserted is determined by the doctype's design.

related tags

- <LINE>
- <CENTER_LINE>

restrictions

Invalid in monospaced examples and math.

Right-adjusted text must fit within the current text margin. If you specify text that is too wide, the text formatter issues a warning message, and you should examine your output.

EXAMPLE

```
<P>Please include the following information:
<RIGHT_LINE>(Name)
<RIGHT_LINE>(Address\smallskip)
<RIGHT_LINE>(Phone Number)
```

This example may produce the following output:

```
Please include the following information:

                                                    Name

                                                 Address
                                            Phone Number
```

<RULE>

Outputs a rule following headings within a table.

FORMAT

<RULE>

ARGUMENTS

None.

related tags

- <TABLE_ROW>
- <TABLE_HEADS>
- <SPAN>
- <TABLE_UNIT_HEADS>

restrictions

Can be used only with an argument to the <TABLE_ROW> , <TABLE_HEADS> or <TABLE_UNIT_HEADS> tags.

Must immediately follow the argument text under which the rule is to be placed.

required terminator

None.

DESCRIPTION

You can specify that a horizontal rule should be included in a table by placing a <RULE> tag inside an argument to one of the following tags:

- <TABLE_ROW>
- <TABLE_HEADS>
- <TABLE_UNIT_HEADS>

The horizontal length of the rule does not correspond to the dimensions of the table. The horizontal length of the rule equals the width of the table column (or the spanned columns, if the argument also is preceded by a <SPAN> tag).

EXAMPLE

See the example in the description of the <TABLE_UNIT> tag.

<S>

Labels the system portion of a dialog between user and system in an interactive example.

FORMAT

<S> *(text)*

ARGUMENTS

text

Specifies the text of the system message.

related tags

- <INTERACTIVE>
- <U>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <S> tag labels the system portion of a dialog between the system and a user. An example containing this type of dialog needs to have both parts identified in order to differentiate the two types of text in the source code, the output, or both.

The <S> and <U> tags are used also to differentiate the system and user text inside of examples created with the <EXAMPLE_SEQUENCE> and <EXI> tags, in the Software Doctype. For more information on this doctype, refer to the *VAX DOCUMENT User Manual, Volume 2.*

EXAMPLES

1

```
<P>The system prompt <S>($) indicates you can enter a command.
```

This example may produce the following output:

The system prompt $ indicates you can enter a command.

2

```
<P>The following example of VAXMAIL contains messages from both
the system and a user of the system:
<INTERACTIVE>
<U>(mail)
<S>(MAIL> )<U>(send)
<S>(To: )<U>(nodename::Courtney)
<S>(%MAIL-E-NOSUCHUSR, no such user COURTNEY at node NODENAME)
<ENDINTERACTIVE>
```

Note that one space is included after a prompt in an <S> tag argument.

This example may produce the following output:

The following example of VAXMAIL contains messages from both the system and a user of the system:

```
mail
MAIL> send
To: nodename::Courtney
%MAIL-E-NOSUCHUSR, no such user COURTNEY at node NODENAME
```

For another example, see the example in the discussion of the <INTERACTIVE> tag.

<SAMPLE_TEXT>

Distinguishes, typographically, an extract of text.

FORMAT

<SAMPLE_TEXT>

ARGUMENTS

None.

related tags

- <CENTER_LINE>

restrictions

Within the sample text, if you want to show a paragraph, you must label it with a <P> tag. The <SAMPLE_TEXT> tag does not provide paragraph spacing.

required terminator

<ENDSAMPLE_TEXT>

DESCRIPTION

The <SAMPLE_TEXT> tag labels text that requires distinctive formatting. The text so labeled will be filled and justified and indented from the normal text margins.

EXAMPLE

```
<P>An evolutionary trend toward niche divergence typically results.
<SAMPLE_TEXT>
<P>Taken from Brown,T.J.: The Skull of the Pronghorn. The McMullen Company,
1970.
<ENDSAMPLE_TEXT>
```

This example may produce the following output:

An evolutionary trend toward niche divergence typically results.

> Taken from Brown,T.J.: The Skull of the Pronghorn. The McMullen Company, 1970.

<SET_APPENDIX_LETTER>

Overrides the default appendix letter assigned to an appendix by VAX DOCUMENT.

FORMAT

<SET_APPENDIX_LETTER> *(appendix-letter)*

ARGUMENTS

appendix-letter

Specifies the letter of the appendix. This argument must be a letter from A to Z.

related tags

- <APPENDIX>
- <SET_CHAPTER_NUMBER>
- The MILSPEC doctype <SET_APPENDIX_NUMBER> tag.

restrictions

This tag should not be used in a file that will be included in a bookbuild or element build. Processing this tag in a bookbuild generates a warning message.

required terminator

None.

DESCRIPTION

Use the <SET_APPENDIX_LETTER> tag to override the default appendix letter created by VAX DOCUMENT. The <SET_APPENDIX_LETTER> tag resets the current appendix letter; if you specified C as the argument, the next <APPENDIX> tag would be Appendix C.

You should place the <SET_APPENDIX_LETTER> tag in your SDML file before the <APPENDIX> tags you want it to affect, because the <SET_APPENDIX_LETTER> tag affects only the <APPENDIX> tags that follow it. The new appendix letter you specify resets the numbering for all following appendixes. For example, if you use the <SET_APPENDIX_LETTER> tag to set the appendix letter to "C," the next appendix will be Appendix C, the appendix following that appendix will be Appendix D, and so on.

<SET_APPENDIX_LETTER> can be used multiple times in an SDML file.

EXAMPLE

In the following example the appendix "Error Messages" is explicitly set to C using the <SET_APPENDIX_LETTER> tag. This will cause any subsequent appendixes to be numbered beginning with the letter D unless another <SET_APPENDIX_LETTER> tag is used to reset the current appendix letter.

```
<SET_APPENDIX_letter>(C)
<APPENDIX>(Error Messages\error_msg_ap)
<p>
The following error messages...
```

<SET_CHAPTER_NUMBER>

Overrides the default chapter number assigned to a chapter by VAX DOCUMENT.

FORMAT

<SET_CHAPTER_NUMBER> *(chapter-number)*

ARGUMENTS

chapter-number

Specifies the number of the chapter. This argument must be a positive integer.

related tags

- <CHAPTER>
- <SET_APPENDIX_LETTER>

restrictions

This tag should not be used in a file that will be included in a bookbuild or element build. Processing this tag in a bookbuild generates a warning message.

required terminator

None.

DESCRIPTION

Use the <SET_CHAPTER_NUMBER> tag to override the default chapter number created by VAX DOCUMENT. The <SET_CHAPTER_NUMBER> tag resets the current chapter number; if you specify 13 as the chapter number, the next <CHAPTER> tag becomes Chapter 13. This numbering also affects all other <CHAPTER> tags in your SDML file so that the subsequent chapter becomes Chapter 14, and so on.

The <SET_CHAPTER_NUMBER> tag should be placed in your SDML file before the <CHAPTER> tags that you want to affect, because the <SET_CHAPTER_NUMBER> tag affects only the <CHAPTER> tags that follow it.

The new chapter number you specify resets the numbering for all following chapters. For example, if you use the <SET_CHAPTER_NUMBER> tag to set the chapter number to 13, the next chapter will be Chapter 13, and the chapter following that chapter will be Chapter 14, and so on.

The <SET_CHAPTER_NUMBER> can be used multiple times in an SDML file.

EXAMPLE

In the following example the chapter is set to 13 using the <SET_CHAPTER_NUMBER> tag. This will cause any subsequent chapters to be numbered beginning with number 14 unless another <SET_CHAPTER_NUMBER> tag is used to reset the current chapter number.

```
<SET_CHAPTER_NUMBER>(13)
<CHAPTER>(Supported Devices\sup_dev_chap)
<p>
The primary supported devices...
```

<SET_CONDITION>

Creates or removes a condition name.

FORMAT

<SET_CONDITION> *(condition-name[\ REMOVE])*

ARGUMENTS

condition-name

Specifies a name used to conditionalize a portion of your SDML file. This name is limited to 28 characters in length.

REMOVE

Causes the condition-name to be removed.

related tags

- <CONDITION>

restrictions

None.

required terminator

None.

DESCRIPTION

The <SET_CONDITION> tag is used to create or remove a condition name. Conditions are specified with the <CONDITION> tag.

For a complete explanation of creating conditional text, refer to Chapter 4.

Normally, a condition name is created at the front of your input and remains in effect for the whole of the input file. In the rare occasion when you want to process a portion of the input without the condition name, it can be removed by repeating the <SET_CONDITION> tag with the REMOVE argument.

You can use the /CONDITION qualifier on the DOCUMENT command line instead of placing the <SET_CONDITION> tag into your file. For example, you might use the command line qualifier "/CONDITION=local" instead of putting " <SET_CONDITION> (local)" in your input file.

EXAMPLE

See the example under the discussion of the <CONDITION> tag.

<SET_FIGURE_FILE_SPACING_DEFAULT>

Overrides the default for vertical blank space that appears before and after the inclusion of a graphics file.

FORMAT

<SET_FIGURE_FILE_SPACING_DEFAULT> *([space-before] \ [space-after])*

ARGUMENTS

space-before

Specifies the amount of vertical space in picas that is to be output before the graphic file (there are 6 picas to an inch). This argument may be a nonnegative integer or decimal number, including zero.

space-after

Specifies the amount of vertical space in picas that is to be output following the graphic file (there are 6 picas to an inch). This argument may be a nonnegative integer or decimal number, including zero.

related tags

- <FIGURE_FILE>

restrictions

None.

required terminator

None.

DESCRIPTION

Each document design specifies default values for spacing that is inserted before and after a graphics file included using the <FIGURE_FILE> tag. These defaults assume that the actual graphic file itself does not have any built-in white space.

In some instances, graphics files may have included white space, and you may not be able to remove it from the graphics file itself. In these instances, you can use the <SET_FIGURE_FILE_SPACING_DEFAULT> tag to modify the amount of white space that VAX DOCUMENT places before and after the graphics file.

The values specified by the <SET_FIGURE_FILE_SPACING_DEFAULT> tag remain in effect for all subsequent figures, or until you use the <SET_FIGURE_FILE_SPACING_DEFAULT> tag again.

If you do not specify one of the arguments to the <SET_FIGURE_FILE_SPACING_DEFAULT> tag, then that spacing remains the same as though the tag had not been used.

EXAMPLES

1
```
<SET_FIGURE_FILE_SPACING_DEFAULT>(0\1.24)
<FIGURE>(Graphic File Inclusion)
<FIGURE_FILE>(LN03\MY_FILES:SANTA.SIX\13.76)
<ENDFIGURE>
```

In this example, the figure file spacing is set to 0 picas before a figure and 1.24 picas after a figure. All subsequent graphics files included using the <FIGURE_FILE> tag will also have this amount of white space around the graphic figure.

2
```
<SET_FIGURE_FILE_SPACING_DEFAULT>(10\0)
<FIGURE>(Graphic File Inclusion)
<FIGURE_FILE>(LN03\MY_FILES:SANTA.SIX\13.76)
<ENDFIGURE>
<SET_FIGURE_FILE_SPACING_DEFAULT>(2\2)
```

In this example, the figure file spacing is set to 10 picas before the figure and 0 picas after it. The use of the second <SET_FIGURE_FILE_SPACING_DEFAULT> tag resets the amount of spacing for any subsequent figures to 2.0 picas before the graphic figure and 2.0 picas after the graphic figure.

<SET_TABLE_ROW_BREAK_DEFAULT>

Overrides the default value for a multipage table's first valid break.

FORMAT

<SET_TABLE_ROW_BREAK_DEFAULT> *(number-of-rows)*

ARGUMENTS

number-of-rows

Specifies the number of rows that the table tags will use as a default when it creates multipage tables. *number-of-rows* must be a positive integer.

You can specify <SET_TABLE_ROW_BREAK_DEFAULT> anywhere in an SDML file. The value you specify using this tag will affect subsequent tables until the next occurrence of the tag or until the end of the file.

related tags

- <TABLE_ROW_BREAK>

restrictions

None.

required terminator

None.

DESCRIPTION

By default, tables are considered "multipage," that is, if there is not enough room on the current page for a table, it will be continued onto subsequent pages, with any captions and headings repeated at the top of each new page. When the text formatter chooses places in the table at which to insert page breaks, it normally breaks the table between <TABLE_ROW> tags. By default, it assumes that it is all right to break a table between any two table rows after the first.

You can override this default behavior in the following ways:

- Specify the <SET_TABLE_ROW_BREAK_DEFAULT> tag to provide a default number of rows that must be on the first page of the table. For example, if you specify <SET_TABLE_ROW_BREAK_DEFAULT> (3), any subsequent table will not be broken until after the third row. (You might want to use this default if your table consists of single-line items, for instance.)
- Specify the table attribute CONTROLLED, which indicates that you are going to specify, explicitly, the range in which the table will be allowed to break. You then must use <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) to indicate the first and last allowable page break points. Between these two tags, the table may be broken between any two <TABLE_ROW> tags.
- Use the KEEP attribute to indicate that the table must not be broken across pages, unless it is longer than a page.
- Use <VALID_TABLE_ROW_BREAK> to indicate a place within a long table row that is an allowable break point.

- Use <NESTED_TABLE_BREAK> to indicate a place in a nested table that is an allowable break point (a nested table is a table that is nested within an argument to another table's <TABLE_ROW>).

EXAMPLE

```
<SET_TABLE_ROW_BREAK_DEFAULT>(5)
<TABLE>
<TABLE_ATTRIBUTES>(SINGLE_SPACED)
<TABLE_SETUP>(2\5)
<TABLE_HEADS>(Code\Numeric Value)
<TABLE_ROW>(A\1)
<TABLE_ROW>(B\2)
    .
    .
    .
<TABLE_ROW>(Z\26)
<ENDTABLE>
```

In this example, a table consists of short, one-line items. The <SET_TABLE_ROW_BREAK_DEFAULT> tag is used to indicate that this table, and any following it, must not be broken until after the fifth row.

<SINGLE_QUOTE>

Outputs, within text, a single quotation mark as it appears on a keyboard.

FORMAT

<SINGLE_QUOTE>

ARGUMENTS

None.

related tags

- <DOUBLE_QUOTE>
- <QUOTE>

restrictions

None.

required terminator

None.

DESCRIPTION

The reason for the <SINGLE_QUOTE> tag lies in a mismatch between conventions used in the typesetting world and conventions used in the computing world. Your terminal keyboard has two possible characters you can use for quotation marks: " (ASCII 34) and ' (ASCII 39). Neither distinguishes between open quotation marks and closed quotation marks. You use the same character twice to enclose a word in quotation marks:

```
"a quoted string"
       or
'a quoted string'
```

Typesetters, on the other hand, scrupulously distinguish between open quotation marks and closed quotation marks. The former look like small sixes; the latter, like small nines. (Actually, in the text font used on the LN01, the open quotation marks look like splinters with their points up; the closed quotation mark looks like splinters with their points down.) Look carefully at any typeset document and you will immediately see the difference between open and closed quotation marks.

To obtain a single quotation mark in your output, looking as it does when shown on the terminal (as an apostrophe), use the <SINGLE_QUOTE> tag. If you simply use the keyboard single quotation mark instead of the tag, your output will contain unmatched closed single quotation marks—one half of what should be a matched pair. (See the example section.) If you use the tag, your output will contain a single quotation mark that looks like the one produced by the terminal keyboard.

EXAMPLE

```
<P>You can cause the translation of a symbol by using a single
quotation mark <PARENDCHAR>(<SINGLE_QUOTE>) directly in front of it.
```

This example shows the use of the <SINGLE_QUOTE> tag. It may produce the following output:

You can cause the translation of a symbol by using a single quotation mark (') directly in front of it.

```
<P>You can cause the translation of a symbol by using a single
quotation mark (') directly in front of it.
```

This example shows what your output would be like without using the tag, but just entering the character from the keyboard.

You can cause the translation of a symbol by using a single quotation mark (') directly in front of it.

<SPAN>

Specifies that the accompanying argument in a table row or a table head should span more than one table column.

FORMAT

<SPAN> *(number-of-columns[\ LEFT])*

ARGUMENTS

number-of-columns

Specifies the number of table columns to be spanned by the text immediately following the <SPAN> tag. This argument must be a whole number and must be no larger than the number of table columns remaining in the table row, including the table column in which the <SPAN> tag itself was placed.

LEFT

Specifies that the text of the argument should be aligned to the left in the left-most of the spanned columns. If this argument is omitted the text of the argument is centered in the spanned columns.

related tags

- <RULE>
- <TABLE_HEADS>
- <TABLE_UNIT_HEADS>

restrictions

Can be used only within an argument to the <TABLE_ROW> , <TABLE_HEADS> , or <TABLE_UNIT_HEADS> tags.

Must immediately precede the argument text.

required terminator

None.

DESCRIPTION

The <SPAN> tag specifies that the argument text that follows the tag should not be confined to a single table column, but should span additional columns. The text is displayed without regard for the gutter that separates the columns. The text is centered in the spanned columns, unless the LEFT argument is supplied.

If the text length exceeds the width of the spanned columns it is broken at a word boundary and displayed on additional table lines. Any additional lines of text are centered or left justified to agree with the alignment of text in the first line.

Use the <RULE> tag to place a rule beneath a spanned heading created using the <SPAN> tag.

EXAMPLES

1

```
<TABLE>
<TABLE_SETUP>(3\12\12)
<TABLE_HEADS>(<SPAN>(3)Types of Ancient Weaponry)
<TABLE_UNIT>
<TABLE_UNIT_HEADS>(Polearms)
<TABLE_ROW>(Spear\Javelin\Halberd)
<ENDTABLE_UNIT>
<TABLE_UNIT>
<TABLE_UNIT_HEADS>(Maces\<SPAN>(2\LEFT)Swords<RULE>)
<TABLE_UNIT_HEADS>( \Short swords\Long swords)
<TABLE_ROW>(Great Mace\Gladius\Great Sword)
<ENDTABLE_UNIT>
<ENDTABLE>
```

This example shows how the <SPAN> tag is used to center the table heading over three columns and also how the LEFT keyword is used to position a table unit heading; note the use of the global <RULE> tag in this example. This example may have the following output:

Types of Ancient Weaponry		
Polearms		
Spear	Javelin	Halberd

Maces	**Swords**	
	Short swords	**Long swords**
Great Mace	Gladius	Great Sword

2

```
<TABLE>
<TABLE_SETUP>(5\9\9\9\9)

<TABLE_ROW>(\<SPAN>(2)Small Fixture\<SPAN>(2)Large Fixture)

<TABLE_ROW>(Component\Minimum\Maximum\Minimum\Maximum)
<ENDTABLE>
```

The output of this example is expected to look approximately like this:

```
                  Small Fixture           Large Fixture
   Component     Minimum    Maximum     Minimum    Maximum
```

However, the <SPAN> tag will actually center "Small Fixture" correctly, but "Large Fixture" may be pushed too far to the right:

	Small Fixture		Large Fixture	
Component	Minimum	Maximum	Minimum	Maximum

This occurs because the <SPAN> tag attempts to span the last column to the margin. To fix this problem, code the table to specify an additional column and specify a width for the next to last column, thereby limiting the distance that is spanned. The following example gives an example of such a correction:

```
<table>
<table_setup>(6\8\8\8\8\8)
<table_row>(\<span>(2)Small Fixture\<span>(2)Large Fixture)
<table_row>(Component\Minimum\Maximum\Minimum\Maximum)
<endtable>
```

Produces:

	Small Fixture		Large Fixture	
Component	Minimum	Maximum	Minimum	Maximum

<SPECIAL_CHAR>

Provides access to special characters that are not available on the terminal keyboard.

FORMAT

<SPECIAL_CHAR> *(keyword)*

ARGUMENTS

keyword

Specifies a keyword associated with the special character you want to produce in your output file. The following are valid keywords:

Keyword	Character
FULL_DIAMOND	♦
TRADEMARK_SYMBOL	™
REGISTERED_SYMBOL	®
SECTION_SIGN	§
DAGGER	†
DOUBLE_DAGGER	‡
OPEN_DOUBLE_BRACKET	⟦
CLOSE_DOUBLE_BRACKET	⟧

related tags

- <MCS>
- <MATH_CHAR>

restrictions

The character must be a non-math character.

required terminator

None.

DESCRIPTION

The <SPECIAL_CHAR> tag provides access to special characters that are not available on the terminal keyboard.

EXAMPLE

```
%COPY-I-COPIED, $DISK1:[MYDIRECTORY]MYFILE.PLI;5<SPECIAL_CHAR>(FULL_DIAMOND)
copied to $DISK1:[YOURDIRECTORY]YOURFILE.PLI;5
```

In this example, the special character ♦ is used in an example of terminal output to show that text wraps at the end of a line. The example may produce the following output:

```
%COPY-I-COPIED, $DISK1:[MYDIRECTORY]MYFILE.PLI;5
♦

copied to $DISK1:[YOURDIRECTORY]YOURFILE.PLI;5
```

<SUBHEADx>

Marks an unnumbered subsidiary heading.

FORMAT

<SUBHEAD1> *(heading-text)*
<SUBHEAD2> *(heading-text)*

ARGUMENTS

heading-text
The text of the subsidiary heading.

related tags

- <HEAD1> through <HEAD6>
- <CHEAD>

restrictions

None.

required terminator

None.

DESCRIPTION

The <SUBHEAD1> and <SUBHEAD2> tags label unnumbered subsidiary headings. Each reflects a logical hierarchy within the structure of the text.

These subheadings are not numbered and do not appear in the table of contents. They cannot be readily used for cross-references and should be used only when the clarity of your exposition requires such a fine level of distinction.

In some book doctypes, the distinction is made between <SUBHEAD1> and <SUBHEAD2> by making <SUBHEAD2> a run-in heading. The doctype design controls the text formatter's placement of the text following the <SUBHEAD2> . Whether or not you expect the heading to be run-in, always tag the text element following the heading (for example, always mark a paragraph with a <P> tag).

EXAMPLE

```
<SUBHEAD1>(Using a Subhead)
<P>The use of subheads should be restricted to occasions when the clarity
of your exposition absolutely requires one.
```

This example might produce output like the following:

Using a Subhead

The use of subheads should be restricted to occasions when the clarity of your exposition absolutely requires one.

<TABLE>

Begins a sequence of columnar data.

FORMAT

<TABLE> *[(table-caption[\ symbol-name])]*

ARGUMENTS

table-caption

Specifies the text of the caption to be associated with the table. The table caption and the associated table number will be written to the table of contents for the document.

Note that the presence of this argument indicates that the table is a formal, numbered table. If the body of the table spans more than a single page of text, the table caption is repeated on each page on which the table continues.

symbol-name

Specifies the symbolic identifier to be associated with the table. The symbol identifier will be assigned a numeric value which will be the current table number. The symbol and its value are placed in the symbol table.

The symbol name can consist of up to 64 alphanumeric characters or underscores. It must not begin with a leading underscore.

related tags

- <TABLE_ATTRIBUTES>
- <TABLE_SETUP>
- <TABLE_HEADS>
- <TABLE_ROW>
- <TABLE_FILE>
- <TABLE_UNIT>
- <TABLE_KEY>
- <TABLE_SPACE>
- <SPAN>
- <RULE>

restrictions

The <TABLE> tag is not valid in monospaced examples, in math, or in figures.

required terminator

<ENDTABLE>

EXAMPLE

```
<P>An expression can evaluate to either an integer or
a string value, depending on the types of values used in the expression
and the operations used to manipulate them.
<REFERENCE>(express_modes_tab) summarizes the rules
for determining the mode of an expression.
<TABLE>(Rules for Determining Expression Modes\express_modes_tab)
<TABLE_ATTRIBUTES>(MULTIPAGE)
<TABLE_SETUP>(2\33)
<TABLE_HEADS>(Expression\Value Type)
 <TABLE_ROW>(Integer value\Integer)
<TABLE_ROW>(String value\String)
<TABLE_ROW>(Integer lexical function\Integer)
<TABLE_ROW_BREAK>(first)
<TABLE_ROW>(String lexical function\String)
<TABLE_ROW>(Integer symbol\Integer)
<TABLE_ROW>(String symbol\String)

<ENDTABLE>
```

This example shows how to produce a two-column table. This example may produce the following output:

An expression can evaluate to either an integer or a string value, depending on the types of values used in the expression and the operations used to manipulate them. Table x—x summarizes the rules for determining the mode of an expression.

Table x—x Rules for Determining Expression Modes

Expression	Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String

<TABLE_ATTRIBUTES >

Specifies special formatting for a table.

FORMAT

<TABLE_ATTRIBUTES> *(keyword-1 [\ keyword-2... \ keyword-5])*

ARGUMENTS

keyword-1...keyword-5

Specifies one to five keywords that indicate special formatting for the table. These keywords are listed in the following table. Note that the MULTIPAGE and KEEP keywords are mutually exclusive and cannot be used together.

Keyword	Description
CONTROLLED	Indicates, for a multipage table, that the table page breaking will be under explicit control of <TABLE_ROW_BREAK> tags. When you use the CONTROLLED keyword, you must specify <TABLE_ROW_BREAK> (FIRST) and <TABLE_ROW_BREAK> (LAST) tags to give the text formatter an allowance for page breaking. This keyword may not be used with the KEEP keyword.
KEEP	Indicates that the table must be kept on one page. If there is not enough room on the current page, a new page is output. If the table exceeds the length of a page, the text formatter issues a warning message and not all the table will be output.
MAXIMUM	Indicates that the text of the table will be adjusted to a smaller point size in order to fit within the left and right margins of the page. Note that if this argument is not specified, and if a table is determined during text formatting to be too wide, the text formatter will adjust the size of the text automatically.
MULTIPAGE	Indicates that the table is either of the following: • Longer than a page • Allowed to break across pages This is the default. If MULTIPAGE is specified, the <TABLE_ROW_BREAK> tag can be used to control page breaks.
SINGLE_SPACED	Specifies that the rows in the table are to be single-spaced; that is, there should be no more vertical space between lines of different table rows than there are between lines within the same table row.
WIDE	Specifies that the width of the table exceeds the document's default width for text. Depending on the document type, this argument may be interpreted as follows: • If the document style contains a left margin area that is normally used for headings, the table's width will span that area as well as the normal text area. • If the document uses a multicolumn format, the table suspends multicolumn output while the table is processed. The table is output, and then multicolumn output is restored.

related tags

- <TABLE>
- <TABLE_SETUP>

restrictions

- Must be in the context of a table.
- Must precede the <TABLE_SETUP> tag.
- WIDE and MAXIMUM might not produce error-free output for monospaced output devices.

required terminator

None.

EXAMPLE

```
<TABLE>(Equivalence Names for Default Process Logical Names\equiv_names_proc_tab)
<TABLE_ATTRIBUTES>(WIDE\KEEP)
<TABLE_SETUP>(4\8\10\12)
<TABLE_HEADS>(Logical Name\Interactive\Batch\Command Procedure)
<TABLE_ROW>(SYS$COMMAND\Terminal\Disk\Terminal)
<TABLE_ROW>(SYS$DISK\Disk\Disk\Disk)
<TABLE_ROW>(SYS$INPUT\Terminal\Disk\Disk)
<TABLE_ROW>(SYS$ERROR\Terminal\Log file\Terminal)
<TABLE_ROW>(SYS$LOGIN\Directory\Directory\Directory)
<TABLE_ROW>(SYS$NET\\\)
<TABLE_ROW>(SYS$OUTPUT\Terminal\Log file\Terminal)
<TABLE_ROW>(SYS$SCRATCH\Directory\Directory\Directory)
<TABLE_ROW>(TT\Terminal\Null device\Terminal)
<ENDTABLE>
```

This example may produce the following output:

Table x—x Equivalence Names for Default Process Logical Names

Logical Name	Interactive	Batch	Command Procedure
SYS$COMMAND	Terminal	Disk	Terminal
SYS$DISK	Disk	Disk	Disk
SYS$INPUT	Terminal	Disk	Disk
SYS$ERROR	Terminal	Log file	Terminal
SYS$LOGIN	Directory	Directory	Directory
SYS$NET			
SYS$OUTPUT	Terminal	Log file	Terminal
SYS$SCRATCH	Directory	Directory	Directory
TT	Terminal	Null device	Terminal

<TABLE_FILE>

Causes a separate file containing a formal table to be included in the SDML input file.

FORMAT

<TABLE_FILE> *(file-spec)*

ARGUMENTS

file-spec

The file specification of the file to be included. If a logical name is specified instead, and the SDML input file is an element of a book, you can define the logical name using an <INCLUDES_FILE> tag in the book's profile. If the source file is not an element of a book, or if the profile does not contain the <INCLUDES_FILE> tag, be sure to define the logical name before processing the file through VAX DOCUMENT.

related tags

- <TABLE>

restrictions

Can be used only in the context of a <TABLE> tag.

The <TABLE_FILE> tag must be placed after the <TABLE> tag.

required terminator

None.

DESCRIPTION

The <TABLE_FILE> tag causes the entire contents of a specified file to be included at this point in the SDML file. (It is identical in action to the <INCLUDE> tag.) The included file should be an SDML file containing a completely coded table.

By keeping a table in a separate SDML file, you can include the table in more than one document, or in chapters of the same document, without having to reproduce the code. In each location, the table can be given a different number.

EXAMPLES

1

```
<TABLE>(Default VMS File Types\Stand_Filetypes_Tab)
<TABLE_ATTRIBUTES>(wide)
<TABLE_FILE>(Standard_Filetypes)
<ENDTABLE>
```

This example shows a wide table that is contained in a separate file. The logical name of the file (Standard_Filetypes) should be recorded in the profile along with the proper file specification for the file. See the next example for a sample profile entry.

It should be noted that this included table file will not process individually. It will only process when included in another SDML file that specifies the <TABLE> and <ENDTABLE> tags.

2
```
<REFERENCE>(command_sum_tab) lists all the commands.
<TABLE_FILE>(endlist.sdml)
```

In this example, the file ENDLIST.SDML may contain all the tags required for the table:

```
<TABLE>(Command Summary\command_sum_tab)
<TABLE_ATTRIBUTES>(Multipage)
<TABLE_SETUP>(2\18)
    .
    .
    .
```

This example assumes that the table is being used in only one document. Placing the table in a separate file allows you to process it individually, or in the context of the file in which it occurs.

<TABLE_HEADS>

Specifies column headings for each column in the table.

FORMAT

<TABLE_HEADS> *(col-heading[\ col-heading...])*

ARGUMENTS

col-heading

Specifies the heading for each column; the first argument is the heading for the first column. Up to nine arguments can be specified, depending on the number of columns in the table.

related tags

- <TABLE_UNIT>
- <TABLE_HEADS>
- <SPAN>
- <RULE>

restrictions

- Only used in the context of a <TABLE> tag.
- Maximum of nine column headings.

required terminator

None.

DESCRIPTION

This tag is not required in a table tags sequence. However, it allows you to specify a heading for each column of a table if headings are needed.

A heading can be any length, and is automatically formatted correctly on one or more lines (see the long heading in the example in <TABLE_ROW>).

Within tables, the <TABLE_HEADS> tag can be used to specify multipage headings for a table. It can also be used to place new headings in the middle of a table. However, they can not be used in a table unit.

EXAMPLE

See the example for the <TABLE_ROW> tag.

<TABLE_KEY>

Begins a key or legend for a table.

FORMAT

<TABLE_KEY>

ARGUMENTS

None.

related tags

- <TABLE_KEYREF>
- <FOOTNOTE>

restrictions

Can be used only within the context of a table.

Must appear immediately after the tag <TABLE_SETUP> , either before or after any <FOOTNOTE> tags.

When a <TABLE_KEY> is used, the number of possible footnotes for the same table is reduced from 12 to 11.

The tags that can be used between <TABLE_KEY> and <ENDTABLE_KEY> are restricted to the <P> , <LIST> and emphasis tags. For example, you cannot use a <TABLE_ROW> tag within the table key.

The <TABLE_KEY> tag is not valid in a nested table. <TABLE_KEY> must follow the <TABLE_SETUP> tag for the outermost table.

required terminator

<ENDTABLE_KEY>

DESCRIPTION

Abbreviations or special terms are often used in a table, either in the column headings, or in the entries of the table. The table may then need a key or legend printed below it that explains the special terms. The <TABLE_KEY> tag begins such a table key.

A table key differs from a table footnote as the table key is not numbered and does not refer to a callout in the table. Only one table key can be declared for a table, whereas up to 12 footnotes are possible.

The table key is declared immediately after the <TABLE_SETUP> tag that begins the table and before the first tag that begins the table rows. This placement is identical to the placement for footnotes in a table. If footnotes are present also, they either can precede or follow the table key declaration.

The table key is printed at the foot of the table, following any footnotes. It is printed only if the <TABLE_KEYREF> tag appears in the table. Place <TABLE_KEYREF> anywhere in the argument list that supplies the column heads. This placement guarantees that the table key is printed at the foot of the table and on each page of a multipage table. (If you want the table key to appear only

on selected pages of a multipage table, place <TABLE_KEYREF> with the table row tags for the selected pages.)

EXAMPLE

```
<table>(Compatability of Lock Modes\lock_tab)
  <table_attributes>(wide\keep)
  <table_setup>(7\10\5\5\5\5\5)

 <table_key>
   <emphasis>(Key to Lock Modes\bold)
   <list>(simple)
      <le>NL---Null lock
      <le>CR---Concurrent read
      <le>CW---Concurrent write
      <le>PR---Protected read
      <le>PW---Protected write
      <le>EX---Exclusive lock
   <ENDLIST>
 <endtable_key>

 <table_heads>(Mode of Requested\
                  <span>(5)Mode of Currently Granted Locks<rule>)
   <table_heads>(Lock<table_keyref>\NL\CR\CW\PR\PW\EX)
   <table_row>(NL\Yes\Yes\Yes\Yes\Yes\Yes)
   <table_row>(CR\Yes\Yes\Yes\Yes\Yes\No)
   <table_row>(CW\Yes\Yes\Yes\No\No\No)
   <table_row>(PR\Yes\Yes\No\Yes\No\No)
   <table_row>(PW\Yes\Yes\No\No\No\No)
   <table_row>(EX\Yes\No\No\No\No\No)

<endtable>
```

This example produces the following output:

Table x—x Compatibility of Lock Modes

Mode of Requested	Mode of Currently Granted Locks					
Lock	**NL**	**CR**	**CW**	**PR**	**PW**	**EX**
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

Key to Lock Modes

NL—Null lock
CR—Concurrent read
CW—Concurrent write
PR—Protected read
PW—Protected write
EX—Exclusive lock

<TABLE_KEYREF>

Specifies that a table key be printed below the table (or portion of the table) in which this tag appears.

FORMAT

<TABLE_KEYREF>

ARGUMENTS

None.

related tags

- <TABLE_KEY>

restrictions

Can be used only within the context of a table in which <TABLE_KEY> is specified.

Invalid in nested tables.

Can only be used at the end of an argument list and cannot be embedded in text; the end of an argument list is indicated by a backslash (\) or a closing parenthesis ( )).

required terminator

None.

DESCRIPTION

The <TABLE_KEYREF> tag specifies that the table key (defined with the <TABLE_KEY> . . . <ENDTABLE_KEY> tags) should be printed at the foot of the table. The table key is printed only if the <TABLE_KEYREF> tag appears in the table.

When the table is a multipage table, the table key can be printed at the foot of any or all portions of the table.

If you want the table key printed at the foot of all portions of a multipage table, place the <TABLE_KEYREF> tag at the end of the argument list to the <TABLE_HEADS> tag that defines the table's column headings. This will cause the table key to be repeated on each page of the table.

If the table key is to be printed only on specified pages of a multipage table, place the <TABLE_KEYREF> tag in the <TABLE_ROW> tags in the reference to the table key.

EXAMPLE

See the example under the discussion of the <TABLE_KEY> tag.

<TABLE_ROW>

Specifies text for each column in the current table.

FORMAT

<TABLE_ROW> *(column-text1[\ column-text2[\ ...]])*

ARGUMENTS

column-text

Specifies text for a single column in the table row. The number of arguments specified to this tag is dependent on the number of columns specified for the table. This value is specified in the <TABLE_SETUP> tag. If the number of arguments to <TABLE_ROW> exceeds the number of columns currently in effect for the table, the excess columns are ignored. If the number of arguments is less than the number of columns currently in effect for the table, unspecified columns are output as blanks.

related tags

- <SPAN>
- <RULE>
- <TABLE_ROW_BREAK>

restrictions

Must be used in the context of a table.

If the text in a single table row column exceeds the depth of an output page, the text formatter issues an error message and terminates processing.

Page breaking of long table rows may be controlled, with additional restrictions, with the <VALID_ROW_BREAK> tag.

required terminator

None.

DESCRIPTION

The <TABLE_ROW> tag specifies text for each column in the current table.

EXAMPLE

```
<REFERENCE>(express_modes_tab) summarizes
the rules for determining the mode of an expression.

<TABLE>(Rules for Determining Expression Modes\express_modes_tab)
  <TABLE_ATTRIBUTES>(MULTIPAGE)
  <TABLE_SETUP>(2\43)
  <TABLE_HEADS>(Expression\Value Type)
  <TABLE_ROW>(Integer value\Integer)
  <TABLE_ROW>(String value\String)
  <TABLE_ROW>(Integer lexical function\Integer)
  <TABLE_ROW>(String lexical function\String)
  <TABLE_ROW>(Integer symbol\Integer)
  <TABLE_ROW>(String symbol\String)
  <TABLE_ROW>(Any value .AND. or .OR. any value\Integer)
  <TABLE_ROW>(Any value\Integer)
  <TABLE_ROW>(Any value\Integer)
<ENDTABLE>
```

This example shows how to produce a two-column table. This example may produce the following output:

Table x—x summarizes the rules for determining the mode of an expression.

Table x—x Rules for Determining Expression Modes

Expression	Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
Any value .AND. or .OR. any value	Integer
Any value	Integer
Any value	Integer

<TABLE_ROW_BREAK>

Specifies the boundaries within which a long table can be broken onto a new page.

FORMAT

<TABLE_ROW_BREAK> ({ *FIRST* / *LAST* })

ARGUMENTS

FIRST
LAST
Specifies the bounds within which a multipage table may be broken. After <TABLE_ROW_BREAK> (FIRST) is specified, any row in the table up to the occurrence of <TABLE_ROW_BREAK> (LAST) is treated as a suitable page-breaking point.

related tags

- <NESTED_TABLE_BREAK>
- <VALID_TABLE_ROW_BREAK>

restrictions

The <TABLE_ROW_BREAK> tag is invalid in a table specified with the KEEP option, or in a table that is specified in an argument to a <TABLE_ROW> tag.

required terminator

None.

DESCRIPTION

The <TABLE_ROW_BREAK> tag provides an allowable place for a long table to be broken across pages.

Tables in VAX DOCUMENT are, by default, multipage tables. If <TABLE_ROW_BREAK> is not specified to provide the text formatter with reasonable guidelines to break tables across pages, the text formatter will nonetheless select page breaks. The page breaks selected will not always be suitable. Therefore, use of the <TABLE_ROW_BREAK> tag is recommended.

By default, the text formatter will not break a page within a table row. With some restrictions, valid page breaks within table rows can be specified using <VALID_TABLE_ROW_BREAK> .

Nested tables (that is, tables that are specified within an argument to a <TABLE_ROW> tag) can, with some restrictions, be given valid page breaks using the <NESTED_TABLE_BREAK> tag.

EXAMPLE

```
<P> An expression can evaluate to either an integer or
a string value, depending on the types of values used in the expression
and the operations used to manipulate them.
Table <REFERENCE>(express_modes_tab) summarizes
the rules for determining the mode of an expression.

<TABLE>(Rules for Determining Expression Modes\express_modes_tab)
  <TABLE_ATTRIBUTES>(MULTIPAGE)
  <TABLE_SETUP>(2\43)
  <TABLE_HEADS>(Expression\Value Type)
  <TABLE_ROW>(Integer value\Integer)
  <TABLE_ROW>(String value\String)
  <TABLE_ROW>(Integer lexical function\Integer)
  <TABLE_ROW>(String lexical function\String)
  <TABLE_ROW>(Integer symbol\Integer)
  <TABLE_ROW>(String symbol\String)
  <TABLE_ROW>(Any value .AND. or .OR. any value\Integer)
  <TABLE_ROW>(Any value\Integer)

<COMMENT>(OK to break after this)
  <TABLE_ROW_BREAK>(first)
  <TABLE_ROW>(Any value\Integer)
  <TABLE_ROW>(Integer value\Integer)
  <TABLE_ROW>(String value\String)
  <TABLE_ROW>(Integer lexical function\Integer)
  <TABLE_ROW>(String lexical function\String)
<COMMENT>(Don't break after this)
  <TABLE_ROW_BREAK>(last)

  <TABLE_ROW>(Integer symbol\Integer)
  <TABLE_ROW>(String symbol\String)
  <TABLE_ROW>(Any value .AND. or .OR. any value\Integer)
  <TABLE_ROW>(Any value\Integer)
  <TABLE_ROW>(Any value\Integer)

<ENDTABLE>
```

This example shows how to produce a two-column table. This example may produce the following output:

An expression can evaluate to either an integer or a string value, depending on the types of values used in the expression and the operations used to manipulate them. Table x—x summarizes the rules for determining the mode of an expression.

Table x—x Rules for Determining Expression Modes

Expression	Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
Any value .AND. or .OR. any value	Integer
Any value	Integer
Any value	Integer

<TABLE_ROW_BREAK>

Table x—x (Cont.) Rules for Determining Expression Modes

Expression	Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
Any value .AND. or .OR. any value	Integer
Any value	Integer
Any value	Integer

<TABLE_SETUP>

Declares the number of columns in a table and the effective (approximate) width values to be assigned to each column.

FORMAT

<TABLE_SETUP> *(number \ $width_1$[. . . \ $width_{number-1}$])*

ARGUMENTS

number

Specifies the number of columns in the table. The argument must be a numeric argument in the range of 2 through 9.

$width_1$
$width_{number-1}$

Specify the approximate widths of columns 1 through *number* – 1. The last column, whose width is unspecified, uses the balance of the right-hand side of the page of output.

The width unit is measured in characters. However, this is an approximante measurement because the width of each character depends on the fonts being used in the doctype. You may need to modify the width argument after processing the table and viewing the output.

related tags

- <TABLE_ATTRIBUTES>

restrictions

- The <TABLE_SETUP> tag must be used in the context of a table.
- If <TABLE_SETUP> is omitted from a sequence of <TABLE> tags, no table will be included in your output.
- The <TABLE_SETUP> tag must be placed after the <TABLE_ATTRIBUTES> tag and before the <TABLE_ROW> tags in the table tag sequence.

required terminator

None.

DESCRIPTION

The arguments to this tag define the number of columns in a multicolumn table and the approximate width of each column. A table can have a maximum of nine columns.

In the width argument, define the width of each column except the last. This last column width is supplied automatically.

Values specified in <TABLE_SETUP> are checked during processing to determine whether the values specified exceed or approach the table's threshold for the given document type. Warning or error messages are issued as appropriate.

If the table-column widths do not allow the table to fit within the page width, the text processor substitutes a smaller point size for the font in which the table is set. It then makes another attempt to fit the table within the text margin.

EXAMPLE

```
<TABLE>(Names of Cheeses from around the world\cheeses_tab)
<TABLE_ATTRIBUTES>(wide\multipage)
<TABLE_SETUP>(4\14\14\14)
<TABLE_HEADS>(Cheese Name\Location of production\Type\Color)
<TABLE_ROW>(Spreadable\Great Britain\Curdled\White)
<TABLE_ROW>(Smooth\Switzerland\Aged and Tasty\Off yellow)
<TABLE_ROW>(Chunky\American\Bland\Yellow)
<ENDTABLE>
```

This example may produce the following output:

Table x—x Names of Cheeses from around the world

Cheese Name	Location of production	Type	Color
Spreadable	Great Britain	Curdled	White
Smooth	Switzerland	Aged and Tasty	Off yellow
Chunky	American	Bland	Yellow

<TABLE_SPACE>

Marks the space required for a table that will be pasted in during final production.

FORMAT

<TABLE_SPACE> (value\text)

ARGUMENTS

value

Specifies the amount of vertical space to be left on the page, specified in picas.

text

Specifies text that describes the status of the table ("To Be Set," an art file number, or some other note). The text will be output in the middle of the space left for the table.

related tags

None.

restrictions

Must be used inside a <TABLE_ROW> tag.

required terminator

None.

DESCRIPTION

The <TABLE_SPACE> tag causes a blank space to be left in a table that will be pasted in by hand during final production. The value in the <TABLE_SPACE> tag should be specified in picas, a scale used by typesetters. There are approximately six picas to the inch. Thus, if the table to be pasted in is four inches high, you should specify 24 picas. If you do not specify a value, a default value of eight picas is used.

If you specify some descriptive text in the second argument, that text is output in the middle of the space left for the table.

EXAMPLE

```
<TABLE>(Script Control Blocks\script_tab)
  <TABLE_SETUP>(2\22)
  <TABLE_HEADS>(Block ID\Layout)
  <TABLE_ROW>(SCBEG; starts the scripting
sequence\<TABLE_SPACE>(4\SCBEG diagram))

<TABLE_ROW>(SCMID; gives the body of the script\<TABLE_SPACE>(4\SCMID
diagram))

<TABLE_ROW>(SCEND; ends the scripting sequence\<TABLE_SPACE>(4\SCEND
diagram))
<ENDTABLE>
```

<TABLE_SPACE>

This example shows how a table is coded with the <TABLE_SPACE> tag. In this example, the <TABLE_SPACE> tags reserve space inside of each table row for three separate diagrams.

Table x—x Script Control Blocks

Block ID	Layout
SCBEG; starts the scripting sequence	SCBEG diagram
SCMID; gives the body of the script	SCMID diagram
SCEND; ends the scripting sequence	SCEND diagram

<TABLE_UNIT>

Begins a portion of a table containing rows that are to be grouped as logical units.

FORMAT

<TABLE_UNIT>

ARGUMENTS

None.

related tags

- <TABLE_UNIT_HEADS>

restrictions

Can be used only within the context of a table established with the <TABLE> tag.

The table must not be declared with the KEEP argument.

If table units are long and you want to allow them to break across pages, you must use the <TABLE_ROW_BREAK> (first) and <TABLE_ROW_BREAK> (last) tags within the bounds of <TABLE_UNIT> and <ENDTABLE_UNIT> .

required terminator

<ENDTABLE_UNIT>

DESCRIPTION

The <TABLE_UNIT> tag and its terminating tag, <ENDTABLE_UNIT> , are used to subdivide a table by grouping some number of rows of the table into a unit. The table unit can be given a heading.

EXAMPLE

```
<Table>(String Passing Techniques Used by the Run-Time Library\Str_Tech_Tab)
<X>(String Passing Techniques Used by the Run-Time Library)<comment>(*)
<TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
<Table_Setup>(4\20\7\10)
<Table_Heads>(\<SPAN>(3)String Descriptor Fields<RULE>)
<Table_Heads>(String Type\Class\Length\Pointer)

<Table_Unit>
<Table_Unit_Heads>(<SPAN>(4\LEFT)Input Argument to Procedures)
<Table_Row>(Input string passed by descriptor\Read\Read\Read)
<Endtable_Unit>

<Table_Unit>
<Table_Unit_Heads>(<SPAN>(4\LEFT)Output Argument from Procedures;
Called Procedure Assumes the Descriptor Class)
<Table_Row>(Output string passed by descriptor,
fixed-length\Ignored\Read\Read)
<Table_Row>(Output string passed by descriptor, dynamic\Ignored\Read, may be
modified\Read, may be modified)
<Endtable_Unit>
```

```
<Table_Unit>
<Table_Unit_Heads>(<SPAN>(4)Output Argument from Procedures, Calling Program
Specifies the Descriptor Class in the Descriptor)
<Table_Row>(Output string, fixed-length ---DSC$K_CLASS = S, Z, A, NCA, SD
\Read\Read\Read)
<Table_Row>(Output string, dynamic ---DSC$K_CLASS_D
\Read\Read, may be modified\Read, may be modified)
<Table_Row>(Output string, varying-length ---DSC$K_CLASS_VS
\Read\MAXSTRLEN is read; CURLEN is modified\Read)
<Endtable_Unit>

<Endtable>
```

This example produces the following output:

Table x – x String Passing Techniques Used by the Run-Time Library

	String Descriptor Fields		
String Type	**Class**	**Length**	**Pointer**
Input Argument to Procedures			
Input string passed by descriptor	Read	Read	Read
Output Argument from Procedures; Called Procedure Assumes the Descriptor Class			
Output string passed by descriptor, fixed-length	Ignored	Read	Read
Output string passed by descriptor, dynamic	Ignored	Read, may be modified	Read, may be modified
Output Argument from Procedures, Calling Program Specifies the Descriptor Class in the Descriptor			
Output string, fixed-length —DSC$K_CLASS = S, Z, A, NCA, SD	Read	Read	Read
Output string, dynamic —DSC$K_CLASS_D	Read	Read, may be modified	Read, may be modified
Output string, varying-length —DSC$K_CLASS_VS	Read	MAXSTRLEN is read; CURLEN is modified	Read

<TABLE_UNIT_HEADS>

Specifies headings to be used for a table unit.

FORMAT

<TABLE_UNIT_HEADS> *(col-heading[\ col-heading...])*

ARGUMENTS

col-heading

Specifies the text for the heading for each column; the first argument is the heading for the first column. Up to nine arguments can be specified, depending on the number of columns in the table.

related tags

- <TABLE_UNIT>
- <SPAN>
- <RULE>

restrictions

Can be used only within the context of a table unit established with the <TABLE_UNIT> tag.

Must immediately follow the <TABLE_UNIT> tag.

required terminator

None.

DESCRIPTION

The <TABLE_UNIT_HEADS> tag specifies column headings for each column of the table. The number of column headings depends on the number of columns in the table, as determined by the tag <TABLE_SETUP> .

Often, the heading for a table unit is used to supply a single heading that spans the columns of the table and serves to label the table unit, rather than the individual columns. In this case, an argument is supplied that begins with the <SPAN> tag.

A null argument leaves the corresponding table column blank.

Notice that more than one <TABLE_UNIT_HEADS> tag can be supplied following a <TABLE_UNIT> tag.

When the heading text for a table column requires multiple lines, you can do the following:

- Supply the text as a long argument to a single <TABLE_UNIT_HEADS> tag. The text is automatically displayed on the required number of lines.
- Supply the text of the heading as shorter arguments to successive <TABLE_UNIT_HEADS> tags. In this way, you can control how the text is displayed on successive lines of the heading.

If a page break occurs in the table unit, all lines of the table unit heading are repeated at the top of the next page.

EXAMPLE

See the example in the description of the <TABLE_UNIT> tag.

<TAG>

Labels a tag.

FORMAT

<TAG> *(tag-name [\ tag-arg [\ tag-arg]])*

ARGUMENTS

tag-name
Specifies the name of the tag.

tag-arg
Specifies arguments to the tag.

related tags

- <LITERAL>

restrictions

Invalid in math.

required terminator

None.

DESCRIPTION

The <TAG> places angle brackets (<...>) around any text and optionally creates an argument for the specified tag. For example, when <TAG> (newtag\new_argument) is placed in an SDML file it produces <NEWTAG> (new_argument) in the output but is not evaluated as a tag.

EXAMPLES

1

```
<P>You use the <TAG>(p) tag to begin a new paragraph.
```

This example may produce the following output:

You use the <P> tag to begin a new paragraph.

2

```
Use <TAG>(code_example\WIDE) if your example
has long lines.
```

This example may produce the following output:

Use <CODE_EXAMPLE> (WIDE) if your example has long lines.

3

```
A writer can use the <tag>(line_art) tag to label a rough sketch. A writer
should not use the <literal><icon><endliteral> tag for that purpose.
```

This example shows the difference in output caused by <TAG> and <LITERAL> : A writer can use the <LINE_ART> tag to label a rough sketch. A writer should not use the <icon> tag for that purpose.

<TITLE>

Labels the title used on either a title page or part page.

FORMAT

<TITLE> *(title-text-1[\ title-text-2[\ title-text-3]])*

ARGUMENTS

title-text-n
Specifies the 1 to 3 lines of text for the title.

related tags

- <TITLE_PAGE>
- <PART_PAGE>

restrictions

May only be used in the context of a <PART_PAGE> tag.

Must be preceded by <TITLE_PAGE> .

Accepts only two title-text arguments when used in the SOFTWARE.BROCHURE doctype.

required terminator

None.

DESCRIPTION

The <TITLE> tag labels the title to appear on a title page or part page. It accepts arguments for one to three lines of title-text in all doctypes except SOFTWARE.BROCHURE.

In the SOFTWARE.BROCHURE doctype, the <TITLE> tag accepts one or two title-text arguments. If only the first argument is specified, that title text is placed at the top of the first page and at the bottom of each successive page; if the optional second argument is also specified, that title text is placed at the bottom of each page. The title text placed at the bottom of the page by the SOFTWARE.BROCHURE <TITLE> tag will be overridden by the text argument of any subsequent <CHAPTER> tags.

EXAMPLE

See the example in the <PART_PAGE> tag description.

<TITLE_PAGE>

Labels the beginning of a title page and enables the title page tags.

FORMAT

<TITLE_PAGE>

ARGUMENTS

None.

related tags

None.

restrictions

This tag must be used in the context of a <FRONT_MATTER> tag.

If you are using a preface, the title page must be terminated before the <PREFACE> tag.

required terminator

<ENDTITLE_PAGE>

DESCRIPTION

The following title page tags are enabled by the <TITLE_PAGE> tag:

- <TITLE>
- <ORDER_NUMBER>
- <ABSTRACT>
- <REVISION_INFO>

EXAMPLE

See the example in the <FRONT_MATTER> tag description.

<U>

Labels the user portion of a dialog between user and system in an interactive example.

FORMAT

<U> *(text)*

ARGUMENTS

text
Specifies the text of the user input.

related tags

- <INTERACTIVE>
- <S>

restrictions

None.

required terminator

None.

DESCRIPTION

The <U> tag labels the user portion of a dialog between the system and a user. An example containing this type of dialog needs to have both parts identified in order to differentiate the two types of text in the source code, the output, or both.

The <U> and <S> tags are also used to differentiate the system and user text inside of examples created with the <EXAMPLE_SEQUENCE> and <EXI> tags, in the SOFTWARE doctype. For more information on this doctype, refer to the *VAX DOCUMENT User Manual, Volume 2*.

EXAMPLES

1

```
<P>The following example of VAXMAIL contains messages from both
the system and a user of the system:
<INTERACTIVE>
<U>(mail)
<S>(MAIL> )<U>(send)
<S>(To: )<U>(nodename::Courtney)
<S>(%MAIL-E-NOSUCHUSR, no such user COURTNEY at node NODENAME)
<ENDINTERACTIVE>
```

This example may produce the following output:

The following example of VAXMAIL contains messages from both the system and a user of the system:

```
mail
MAIL> send
To: nodename::Courtney
%MAIL-E-NOSUCHUSR, no such user COURTNEY at node NODENAME
```

<UNDERLINE>

Marks a portion of text to be underlined.

FORMAT

<UNDERLINE> *(text)*
or

<UNDERLINE>
text

.
.
.

<ENDUNDERLINE>

ARGUMENTS

text
Specifies the text string to be underlined.

related tags

None.

restrictions

Because the text string that is marked for underlining will be kept on the same line in the output, do not make the underline longer than the page width or errors might be generated.

required terminator

<ENDUNDERLINE> — Required if the text is not passed as an argument.

EXAMPLE

```
<P>If you use the appropriate flags, your output
may look like this:
<P>
<SAMPLE_TEXT>
<UNDERLINE>(The Decline and Fall of the Roman Empire)
<ENDSAMPLE_TEXT>
```

This example may produce the following output:

If you use the appropriate flags, your output may look like this:

<u>The Decline and Fall of the Roman Empire</u>

<UPDATE_RANGE>

Marks the location at which a new section of updated pages begins.

FORMAT

<UPDATE_RANGE> *(start-page \ end-page)*

ARGUMENTS

start-page

Specifies the page number from the printed documentation of the first page that must be printed. This must be an odd-numbered page.

end-page

Specifies either the last page to be included in a set of update pages or the keyword EOF. If end-page is a number, that number must be an even number. The text formatter output routines that handle updates will automatically generate point-numbered pages if text within the bounds of start-page and end-page will not fit within those pages. If EOF is specified, then the update range continues to the end of a chapter or section.

related tags

- <REVISION>
- <MARK>

restrictions

Must be used in the context of the <REVISION> tag, and <REVISION> must specify the keyword UPDATE. If the <REVISION> tag is not specified, then the <UPDATE_RANGE> and <ENDUPDATE_RANGE> tags are nonoperational; that is, no output will be generated from these tags when the file is processed.

required terminator

<ENDUPDATE_RANGE>

DESCRIPTION

Files coded with the <REVISION> (UPDATE*update-level*) tag and containing <UPDATE_RANGE> tags will process as follows:

- When the text formatter processes a file that has been marked as an update, it processes all text and commands in the file, but does not produce actual output for the DVI file except for those pages marked within an update range.
- When the text formatter reaches the beginning of an update range, it sets the page number to the page number specified as the start of the update. Therefore, it is not important that previous versions of the text formatter file were modified for pagination during final production. All pages processed outside of an update range are not output.

The following rules apply to the placement of the <UPDATE_RANGE> tag:

- If the text on the first update page is in the middle of a text element (for example, a paragraph, a list element, a code example), then the <UPDATE_RANGE> tag must precede the word of text that is the first word on the page.
- If the text on the first update page represents the beginning of a new text element (<P> , <LIST> , <LE> , <CODE_EXAMPLE> , and so on), the <UPDATE_RANGE> tag must be placed immediately preceding the tag for the text element.
- If an update range begins on a page that starts with a continued table, the file must specify the start of the update range on the odd-numbered page preceding the beginning of that table, or (if the table begins on an odd-numbered page) the page on which the table begins. If these pages are not to be a part of the update, they may be discarded.
- If one or more pages before an update range begins contains a floating figure or example, the <FIGURE_ATTRIBUTES> or <EXAMPLE_ATTRIBUTES> tags must be modified to specify KEEP. This will prevent the text formatter from floating the figure or example to the top of the first update page.

  Put a <COMMENT> tag in the file to indicate that the modification was made for the purposes of the update only. For example:

  ```
  <COMMENT>(KEEP added to example for update only...)
  ```

  When the file is subsequently revised, the KEEP arguments can be removed.
- When a file that contains the <REVISION> (UPDATE) tag is processed, the table of contents and index are handled as follows:
  - If the <CONTENTS_FILE> or <INDEX_FILE> tag occurs between the bounds of <UPDATE_RANGE> and <ENDUPDATE_RANGE> , and /CONTENTS or /INDEX is specified on the command line, the contents file or index file is included within the pages for the update range.
  - If the <CONTENTS_FILE> or <INDEX_FILE> tag occurs in the file but is specified outside the bounds of <UPDATE_RANGE> and <ENDUPDATE_RANGE> tags, and if /CONTENTS or /INDEX is specified on the command line, the table of contents file or index file is included in the output file.

EXAMPLE

```
<REVISION>(UPDATE\July 1986)
    .
    .
    .
<UPDATE_RANGE>(5\24)
<P>
The first sentence on page 5 goes here.
    .
    .
    .
20 or more pages of modified text goes here.
    .
    .
    .
The last sentence on page 24.n goes here.
<ENDUPDATE_RANGE>
```

In this example, the updated material begins on page 5 and continues through page 24. When page 24 is reached, the page numbering becomes 24.1, 24.2, etc., until the end of the update range is reached. The <ENDUPDATE_RANGE> tag must be placed in the source file at the position corresponding to the place at which an update sequence ends.

<UPPERCASE>

Labels text that should appear as uppercase in the final output.

FORMAT

<UPPERCASE> *(text)*

ARGUMENTS

text
Specifies the text to appear in uppercase.

related tags

- <LOWERCASE>

restrictions

None.

required terminator

None.

DESCRIPTION

If your book contains a text element, such as a heading, that normally appears in lowercase, you may encounter a situation where you need to overcome the default case in one of your tags and ensure that the result in the final output appears in uppercase. The <UPPERCASE> tag allows you to do this.

EXAMPLE

```
<HEAD2>(here is an example of <UPPERCASE>(uppercase) text)
```

In this example, assume that the doctype being used causes the tag <HEAD2> to output a heading that is in lowercase, no matter what the case of the text passed to it. The <UPPERCASE> tag overrides the default in this <HEAD2> tag. This example produces the following output:

here is an example of UPPERCASE text

<USER_I_MESSAGE>

Sends an informational message to the terminal or log file during processing of a file.

FORMAT

<USER_I_MESSAGE> *(text)*

ARGUMENTS

text

Specifies the text you wish to appear on the terminal or in the log file.

related tags

- <USER_W_MESSAGE>

restrictions

You must process the file with the /LOG qualifier in order to see any <USER_I_MESSAGE> messages.

The message text is limited to 150 characters.

required terminator

None.

DESCRIPTION

The <USER_I_MESSAGE> tag generates a message when the file is processed, but only when the /LOG qualifier is specified on the command line. The message is sent to the terminal if the file is processed interactively or to the log file if it is processed as a batch job.

The messages are listed in the .LIS file if you specify the /LIST qualifier on the command line.

You can use the tag to broadcast any important information that should be noted or flagged during tag translation; for example, you might use it as a reminder that a part of the file is incomplete.

The tag translator displays the line number and the file name in which the tag appears. The message is output in the following format:

```
%TAG-I-USER_IMSG, mmmmmmmmmmmmmmmmmmmm
   Line is nnn of file ffffffffffff
```

Note: **To receive any messages tagged with <USER_I_MESSAGE> tags, you must process the file with the /LOG qualifier on the command line. If you want to generate only a few messages, you might choose to use the <USER_W_MESSAGE> tag and not process the file with the additional qualifier.**

EXAMPLE

```
<USER_I_MESSAGE>(Section 2 is incomplete and requires
information from Tom Jones.)
```

This example shows how a <USER_I_MESSAGE> tag can be used to flag a section of a file that requires further work. The message would only be sent to the terminal if the user processed the file with the /LOG qualifier on the command line. If the text element containing this tag were processed as a batch job, the log file would contain the following entry:

```
%TAG-I-USER_IMSG, Section 2 is incomplete and requires information
from Tom Jones.
   Line is 68 of file part2.gnc
```

<USER_W_MESSAGE>

Sends a warning message to the terminal or log file during processing of a file.

FORMAT

<USER_W_MESSAGE> *(warning-text)*

ARGUMENTS

warning-text
The text you wish to appear on the terminal or in the log file.

related tags

- <USER_I_MESSAGE>

restrictions

The message text is limited to 150 characters.

required terminator

None.

DESCRIPTION

The <USER_W_MESSAGE> tag generates a message when the file is processed, sending it to the terminal if processed interactively or the log file if processed as a batch job.

Note: **This type of message is counted by the tag translator as a warning message. After thirty warning messages, tag translation halts. Therefore, use the <USER_W_MESSAGE> tag for generating messages only if you are issuing few messages. For a file that contains many messages, tag messages with <USER_I_MESSAGE> tags and process the file with the /LOG qualifier.**

You can use it to flag any important information that should be noted during processing; for example, you might use it as a reminder that a part of the file is incomplete. Whatever message you enter as an argument to the <USER_W_MESSAGE> tag is written to the terminal or to the batch log file when the text element that contains it is processed through the tag translator.

The tag translator displays the line number and the file name in which the <USER_W_MESSAGE> tag appears. The message is output in the following format:

```
%TAG-W-USER_WMSG, at tag <USER_W_MESSAGE> on line nnn of file
   ffffffff
   mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

The messages are listed in the .LIS file if you specify the /LIST qualifier on the command line.

EXAMPLE

```
<USER_W_MESSAGE>(Reviewers: Please note missing parameters here.)
```

This example shows how the <USER_W_MESSAGE> tag is used to identify a notice in a file. If the file containing this message was called "Reviewers_copy.SDML," and it was processed as a batch job, the log file would contain the following entry:

```
%TAG-W-USER_WMSG, at tag USER_W_MESSAGE on line nn of file
 Reviewers_copy.SDML.
 Reviewers: Please note missing parameters here.
```

<VALID_BREAK>

Labels a permissible page break within a monospaced example.

FORMAT

<VALID_BREAK>

ARGUMENTS

None.

related tags

- <CODE_EXAMPLE>
- <DISPLAY>
- <INTERACTIVE>
- <LINE_ART>

restrictions

This tag can be used only within monospaced examples created using the <CODE_EXAMPLE> , <DISPLAY> , <INTERACTIVE> , or <LINE_ART> tags.

required terminator

None.

DESCRIPTION

The text formatter attempts to keep an example together on a single page. If there is not enough room for an example on the current page, the text formatter chooses page breaks using blank lines in the example as "good" places to break. If your example contains no blank lines, or if you want to specify better breaking points, you can use <VALID_BREAK> to specify the places that are acceptable page breaks.

EXAMPLE

```
<INTERACTIVE>

<S>($)
<U>(@SYS$SYSTEM:SHUTDOWN)

<S>(        SHUTDOWN -- Perform an Orderly System Shutdown)
<ELLIPSIS>
<S>(CENTRAL, PRINTER, TAPES, DISKS, DEVICES, CARDS, NETWORK, OPER1)
<S>(OPER3, OPER4, OPER5, OPER6, OPER7, OPER8, OPER9, OPER10, OPER11,)
<S>(OPER12)
<VALID_BREAK>

<S>(%SHUTDOWN-I-DISLOGINS, Interactive logins will now be disabled.)
<S>(%SET-I-INTSET, login interactive limit = 0 current interactive value = 17)
<S>(%SHUTDOWN-I-SHUTNET, The DECnet network will now be shut down.)
<S>(%SHUTDOWN-I-STOPQUEMAN, The queue manager will now be stopped.)
<ENDINTERACTIVE>
```

This example shows the use of the <VALID_BREAK> tag.

<VALID_TABLE_ROW_BREAK>

Marks a permissible place that a first-level table row may be broken across pages.

FORMAT

<VALID_TABLE_ROW_BREAK>

ARGUMENTS

None.

related tags

- <NESTED_TABLE_BREAK>

restrictions

This tag must be used in the context of a table. May only be used in the last column of a 2- or 3-column, first-level table. That is, this tag may not be specified in a table that is specified inside a <TABLE_ROW> tag.

required terminator

None.

DESCRIPTION

The <VALID_TABLE_ROW_BREAK> tag provides an allowable place for a long table column to be broken across a page. If you do not use the <VALID_TABLE_ROW_BREAK> tag inside a long table row, the text formatter tries to keep all the text on the same page of output and might issue a PAGE-TOO-LONG error message. In extreme cases, the text formatter could run out of memory trying to process the table row and terminate processing.

EXAMPLE

```
<TABLE>
<TABLE_ATTRIBUTES>(MULTIPAGE)
<TABLE_SETUP>(2\9)
<TABLE_ROW>(Item\Text that goes on for several paragraphs.
<VALID_TABLE_ROW_BREAK>
Text that is still inside the first table row.
<P>
<VALID_TABLE_ROW_BREAK>
The last paragraphs for this table row.)
<ENDTABLE>
```

This example shows the use of the <VALID_TABLE_ROW_BREAK> tag in a table with a long row.

<VARIABLE>

Labels a program variable or number.

FORMAT

<VARIABLE> *(variable-name)*

ARGUMENTS

variable-name
Specifies the name of the variable to be typographically distinguished.

related tags

- <KEYWORD>

restrictions

None.

required terminator

None.

DESCRIPTION

The <VARIABLE> tag names a variable discussed in text. Use of the <VARIABLE> tag and its result in formatted text must be agreed upon by writer, editor, and book designer. Above all, you should seek a consistent usage within a document and across a document set.

EXAMPLE

```
<P>At this point in the processing,
 <VARIABLE>(NUMBER_OF_JELLYBEANS) has the
value
of 2.
```

This example may produce the following output:

At this point in the processing, *NUMBER_OF_JELLYBEANS* has the value of 2.

<VBAR>

Labels an occurrence of a vertical bar in an argument to a tag.

FORMAT

<VBAR>

ARGUMENTS

None.

related tags

- The following tags label other characters that must be tagged when they occur in an argument to a global tag:

  <AMPERSAND>
  <BACKSLASH>
  <CPAREN>
  <OPAREN>

restrictions

Can only be used within an argument to a tag.

required terminator

None.

DESCRIPTION

The tag translator uses a vertical bar (|) to begin a quoted string. An ampersand concludes a quoted string. If you use a literal vertical bar within an argument to an SDML tag, the tag translator reads the vertical bar as beginning a section of text it should treat literally. The vertical bar may prevent the tag translator from evaluating a tag when it should and may cause an error in your output.

To process a vertical bar in an argument to a tag (through to your output), use the <VBAR> tag.

EXAMPLE

```
<SUBHEAD1>(Labeling the Vertical Bar (<VBAR>) Within Your
Code)
<P>To pass a vertical bar (|) in an argument to a tag through
to your output, . . .
```

This example produces the following output:

Labeling the Vertical Bar (|) Within Your Code

To pass a vertical bar (|) in an argument to a tag through to your output, . . .

<X>

Creates an index entry with a reference to the page on which this tag appears.

FORMAT

<X> (index-entry[\ attribute])

ARGUMENTS

index-entry

Specifies the index entry to appear in the index. The capitalization you use in this string will be the capitalization that appears in the index.

Use the <XSUBENTRY> tag to separate the main entry from the first subentry, if used, and the second subentry from the first, as follows:

<X> (main entry <XSUBENTRY> subentry <XSUBENTRY> subentry)

You should use no more than two levels of subentries in a book index.

attribute

Specifies the attributes that control the sorting and formatting of the index entry. You can specify a maximum of five attribute arguments from among the following possibilities. Each attribute must be passed as a separate argument.

Attribute	Function
<XAPPEND> (string)	Causes the indexing software to append the specified string to the page reference in the index.
BEGIN	Causes the indexing program to begin a page-range reference. When you use this attribute, you must pair it with a following <X> tag that has identical text and the END attribute.
BOLD	Causes the page reference number to appear in boldface. This is distinct from any boldfacing that appears in the text of the index entry itself.
END	Causes the index program to end a page-range reference. When you use this attribute, you must pair it with a previous <X> tag that has identical text and the BEGIN attribute.
ITALIC	Causes page reference numbers to appear in italic type in your index entry. If both BOLD and ITALIC are specified, the entry is output in bold italic type. Entries that use the ITALIC attribute are sorted as being distinct from entries that are output in an italic type face using some other means (such as the global <EMPHASIS> tag).

Attribute	Function
MASTER	Causes the index program to insert the entry only in the master index. By default, the entries are inserted only in the local index. This attribute allows you to specify different entries for a book's local index and for the master index of the document set.
<XSORT> (string)	Causes the index program to use the specified string as the sort key when placing this entry in the index. Use this attribute to override the sorting algorithm of the index utility for single index entries (for example, to force an entry with a leading nonalphanumeric character to the top of the index).

related tags

- <Y> —Used for entering a cross-reference index entry

restrictions

Do not place <X> tags within any kind of example. Doing so will interfere with the formatting of the example.

Be sure <X> tags follow headings, commands, and other major text elements that are likely to begin a new page.

In tables, place <X> tags directly next to the items to be indexed. In doing so, the <X> tag will be included within the argument string for table tags such as <TABLE_ROW> or <TABLE_HEADS> .

required terminator

None.

DESCRIPTION

The <X> tag creates an entry in the index to the book. The entry is composed of the main entry and optional subentries specified in the first argument to the tag.

You can control how the index entry is sorted and how it appears by specifying particular attributes.

For instance, the <XAPPEND> attribute is useful when appending text to the end of a page reference, as when referring to an example (5-4ex) or to a table (9-8tab). The <XAPPEND> attribute can also be used for such common indexing terms as "page 3ff" or "5-3 to 5-6 passim."

The appended string is boldfaced or italicized if its page reference is boldfaced or italicized. BOLD and ITALIC can be used together.

You can create an index entry for several inclusive pages by using the BEGIN and END attributes. When using a BEGIN and END pair, you must be certain that the index entries are identical and that any other attributes you specify are the same for each entry, as well.

You can specify that an index entry appear in a master index by using the MASTER attribute. By default, an entry appears in the master index only if this attribute has been specified. It is possible to override this arrangement at the time the master index is built and include every entry in the local index in the master. No entry marked with the MASTER attribute appears in the local index.

If you have an index entry for which you want to control the sorting, you can do so by using the <XSORT> attribute. For example, since leading nonalphabetic characters are ignored by default, the main entry $SEARCH normally appears with other entries beginning with the letter "S." To force the indexing software to place $SEARCH before the "A" section, use the <XSORT> attribute as follows:

```
<X>($SEARCH\<XSORT>($))
```

You must use the <XSORT> attribute if your main entry begins with a double backslash or a right angle bracket.

The <XSUBENTRY> Tag

You can use the <XSUBENTRY> tag within this argument to separate the main entry from the first subentry, and the second subentry from the first, as follows:

<X> (Main entry <xsubentry> subentry-1 <xsubentry> subentry-2)

The <XSUBENTRY> tag can be abbreviated as <XS> . Using this shorter form of the tag, the previous example could be expressed as follows:

<X> (Main entry <xs> subentry-1 <xs> subentry-2)

You can use no more than three levels of subentries in an index.

EXAMPLES

1
```
<X>(File structure\begin)
  .
  .
  .
<X>(File structure\end)
```

These index tags create an inclusive page entry, and may produce the following output.

File structure, 5-4 to 5-7

2
```
<X>(File structure\begin)
  .
  .
  .
<X>(File structure\end\<XAPPEND>( passim))
```

This index tag creates an index entry that may produce the following output.

File structure, 5-4 to 5-7 passim

3

```
<TABLE_ROW>(apples\oranges\pears)
<TABLE_ROW>(<X>(Tropical fruits<XS>bananas)bananas\pineapples\mangos)
<TABLE_ROW>(blackberries\blueberries\strawberries)
```

The <X> tag in this example creates an index entry that will correctly specify the page that the text "bananas" appears on, no matter where VAX DOCUMENT chooses to break the table that contains "bananas." This index entry may produce the following output.

Tropical fruits

bananas, 3-21

<Y>

Creates an index entry with no reference to the page on which this tag appears. Used for cross-references ("See" or "See also" entries).

FORMAT

<Y> (index-entry[\ attribute])

ARGUMENTS

index-entry

Specifies the string that supplies additional information to the main entry and subentries that will appear in the index. The capitalization you use in this string will be the capitalization that appears in the index.

Use the <XSUBENTRY> tag to separate the main entry from the first subentry, and the second subentry from the first, as follows:

Main entry <xsubentry> subentry

You should use no more than two levels of subentries in a book index.

attribute

Specifies the attributes that control the sorting and formatting of the index entry. You can specify a maximum of two attribute arguments from among the following possibilities. Each attribute must be passed as a separate argument.

Attribute	Function
MASTER	Causes the index program to insert the entry only in the master index. By default, the entry is inserted only in the local index. This attribute allows you to specify different entries for a book's local index and for the master index of the document set.
<XSORT> (string)	Causes the index program to use the specified string as the sort key when placing this entry in the index. You must enclose the string in either double quotation marks (") or single quotation marks ('). Use this attribute to override the sorting algorithm of the index utility for single index entries.

related tags

- <X>

restrictions

Do not place <Y> tags within any kind of example. Doing so will interfere with the formatting of the examples.

required terminator

None.

DESCRIPTION

The <Y> tag creates an unpaged entry in the index to the book. The entry that appears in the index is composed of the main entry and "See" or "See also" subentries specified in the first argument to the tag.

You can control how the index entry is sorted and where it will appear by specifying particular attributes in arguments two through four of the <Y> tag.

You can specify that an index entry appear in a master index by using the MASTER attribute. By default, an entry will appear in the master index only if this attribute has been specified. It is possible to override this arrangement and include every entry in the local index in the master.

If you have an index entry for which you want to control the sorting, you can do so by using the <XSORT> attribute. For example, because leading nonalphabetic characters are ignored by default, the main entry $SEARCH normally appears with other entries beginning with the letter "S." To force the indexing software to place $SEARCH before the "A" section, you can use the SORT attribute as follows:

```
<Y> ($SEARCH <XS> See System Services\ <XSORT> ($))
```

EXAMPLE

```
<X>(File structure designators)
<Y>(File structure designators<XSUBENTRY>See also Header blocks)
```

This index command would create the following index entry, assuming these tags appeared on page 1-3:

```
File stucture designators, 1-3
    See also Header blocks
```

A VAX DOCUMENT Command Summary

This appendix describes the DCL command line interface to VAX DOCUMENT. A summary of this information can be obtained by typing the following command once VAX DOCUMENT is installed:

```
$ HELP DOCUMENT
```

See Chapter 4 for tutorial information on how to process your files using the DOCUMENT command.

DOCUMENT

Invokes the VAX DOCUMENT document production system.

FORMAT

DOCUMENT *input-file-spec doctype destination*

Command Qualifier	Default
/[NO]BATCH[=qualifier-keyword]	*/NOBATCH*
/CONDITION=condition-name	*None.*
/[NO]CONTENTS	*/NOCONTENTS*
/[NO]DEVICE_CONVERTER[=device-keyword]	*/DEVICE_CONVERTER*
/[NO]DIAGNOSTICS[=file-spec]	*/NODIAGNOSTICS*
/ELEMENT=file-spec	*None.*
/INCLUDE=file-spec	*None.*
/[NO]INDEX[=index-keyword]	*/NOINDEX*
/[NO]KEEP[=filetype-keyword]	*/NOKEEP*
/[NO]LIST[=file-spec]	*/NOLIST*
/[NO]LOG	*/LOG*
/[NO]MAP[=file-spec]	*/NOMAP*
/[NO]MASTER_INDEX[=index-keyword]	*/NOMASTER_INDEX*
/OUTPUT=file-spec	*See text.*
/[NO]PRINT[=qualifier-keyword]	*/PRINT*
/PROFILE=file-spec	*None.*
/[NO]SYMBOLS=file-spec	*/NOSYMBOLS*
/[NO]TAG_TRANSLATOR	*/TAG_TRANSLATOR*
/[NO]TEXT_FORMATTER	*/TEXT_FORMATTER*

restrictions

None.

PARAMETERS

input-file-spec

Specifies the input file to be processed. Note that wildcards are not allowed in the input file specification.

The default file type of the input file specification expected by VAX DOCUMENT is SDML. If qualifiers are specified, VAX DOCUMENT determines the default file type based on the qualifiers and the destination keyword you specify.

Table A–1 lists the default file types and the qualifiers and default destinations that provide them.

Table A–1 Default File Types

VAX DOCUMENT Qualifiers Used	Default Destination Keyword Used	Default Input File Type
None.	Any	SDML
/TAG_TRANSLATOR	Any	SDML
{ /NOTAG_TRANSLATOR /TEXT_FORMATTER }[1]	Any	TEX
{ /NOTAG_TRANSLATOR /NOTEXT_FORMATTER /DEVICE_CONVERTER }	LN03	DVI_LN03
	PS	DVI_PS
	LINE	DVI_LINE
	TERMINAL	DVI_LINE
	MAIL	DVI_LINE
{ /NOTAG_TRANSLATOR /NOTEXT_FORMATTER /NODEVICE_CONVERTER /PRINT }	LN03	LN03
	PS	PS
	LINE	LINE
	TERMINAL	TERM
	MAIL	TXT

[1]Braces in this table indicate that the enclosed qualifiers occur together on the command line.

doctype

Specifies the type of document being produced. The *doctype* you select on the command line determines the style of your output and determines the SDML tags you can use. Note that some tags are valid only in specific doctypes. See the *VAX DOCUMENT User Manual, Volume 2,* for more information on doctype-specific tags.

All doctypes have a default design. Some doctypes have several designs. These doctypes accept a *design* keyword as part of the doctype specification. The design keyword specifies an alternate design within the doctype. For example, the SOFTWARE doctype has designs for different page sizes, heading level formats, and so on.

The design keyword is specified as part of the doctype and is separated from the doctype keyword by a period (.). For example, the REPORT doctype has two alternate designs: one in which the text runs the full text page width (the default), and another in which the text is placed in two columns. You specify the default design as REPORT and the two-column design as REPORT.TWOCOL. You can abbreviate the doctype keyword to any unique string. For example, you could abbreviate REPORT.TWOCOL to REP.TWO, or even R.T as long as each keyword is unique.

Table A–2 summarizes the supported VAX DOCUMENT doctype keywords and includes a basic use for each keyword, and a summary description of the available designs. See the *VAX DOCUMENT Design Samples* manual for more information on each of these doctype designs.

Table A–2 VAX DOCUMENT Doctypes

Doctype Keyword	Used to Create	Design Description
ARTICLE	Articles	$8\frac{1}{2} \times 11$ inches, numbered or unnumbered headings, text is placed in a two-column format
LETTER	Letters or memos	$8\frac{1}{2} \times 11$ inches, unnumbered headings
MANUAL.GUIDE	User manuals	7×9 inches, numbered headings
MANUAL.PRIMER		7×9 inches, unnumbered headings
MANUAL.REFERENCE		$8\frac{1}{2} \times 11$ inches, numbered headings
MILSPEC	Military specifications	$8\frac{1}{2} \times 11$ inches, numbered headings
OVERHEADS	Overhead slides for transparencies	$8\frac{1}{2} \times 11$ inches, no headings
OVERHEADS.35MM		$6\frac{1}{2} \times 5\frac{1}{2}$ inches, no headings
REPORT	General-purpose documents	$8\frac{1}{2} \times 11$ inches, numbered headings
REPORT.TWOCOL		$8\frac{1}{2} \times 11$ inches, numbered headings, text is placed in a two-column format
SOFTWARE.BROCHURE	User manuals containing detailed information on software	7×9 inches, unnumbered headings
SOFTWARE.GUIDE		7×9 inches, numbered headings
SOFTWARE.HANDBOOK		7×9 inches, numbered headings
SOFTWARE.POCKET_REFERENCE		$5\frac{1}{2} \times 7$ inches, numbered headings
SOFTWARE.REFERENCE		$8\frac{1}{2} \times 11$ inches, numbered headings
SOFTWARE.SPECIFICATION		$8\frac{1}{2} \times 11$ inches, numbered headings

You can have additional local doctype keywords defined at your site. See your VAX DOCUMENT system administrator for information on local doctype keywords.

destination

Specifies the output device *destination* keyword for the document. VAX DOCUMENT supports the following output destination keywords (these keywords are the default keywords provided when VAX DOCUMENT is installed, your local destination keywords may be different):

Destination Keyword	Formatted For
LINE [1]	A line printer
LN03 [1]	An LN03 laser printer
PS[1]	Any Digital-supported POSTSCRIPT output device, such as the PRINTSERVER 40 or the LN03R SCRIPTPRINTER
TERMINAL	A standard ANSI terminal, such as the VT-100
MAIL	Sending through the VMS Mail Utility

[1]These destinations are installed at your site only if they were selected during the installation procedure.

You can abbreviate the destination keyword to any unique string. For example, you could abbreviate LN03 to LN0, or even LN as long as that string is unique.

You can have additional local destination keywords defined at your site. See your VAX DOCUMENT system administrator for information on local destination keywords.

DESCRIPTION

DOCUMENT is the command you specify to invoke VAX DOCUMENT. VAX DOCUMENT is a document production system that lets you create documents in many different formats for a variety of output devices. All documents are created by entering *SDML tags* into an input file which is then processed by VAX DOCUMENT.

The DOCUMENT command requires three parameters:

- Input File Specification

  Specifies the input file for VAX DOCUMENT. This file is by default an SDML file containing SDML tags, however, it can also be one of the intermediate files generated by VAX DOCUMENT.

- Doctype

  Specifies the document type keyword for which the input file should be processed. This keyword specifies the kind of document to be created (a letter, a software manual, a journal article, and so on).

- Destination

  Specifies the final processing destination for the input file. This keyword typically specifies a format used by a printer, but can specify formats for terminals or the VMS Mail Utility.

You can use the qualifiers to the DOCUMENT command to create indexes, master indexes, tables of contents, and to modify the default processing of your input file.

COMMAND QUALIFIERS

/BATCH[=(qualifier-keyword[,qualifier-keyword...])] /NOBATCH

Specifies whether VAX DOCUMENT should be run interactively or submitted as a batch job. The default qualifier is /NOBATCH, which specifies that VAX DOCUMENT should be run interactively.

The /BATCH qualifier builds and issues a DCL SUBMIT command that submits a job to SYS$BATCH with a job name that has the same file name as the *input-file-spec* used on the command line, prefixed with the string "DOC$." For example, the file ROUTINES.SDML would be submitted as the job DOC$ROUTINES.

You can use any of the DCL SUBMIT command qualifiers with VAX DOCUMENT by passing these qualifiers as keywords to the /BATCH qualifier. For example, if you want a command procedure to be run after 9:00 and want to be notified when it completes, you could use the following command:

```
$ SUBMIT somefile.com /AFTER=09:00/NOTIFY
```

You can specify the same options for your document processing using the /BATCH qualifier on the command line as follows:

```
$ DOCUMENT somefile /BATCH=(AFTER=09:00,NOTIFY) LETTER LN03
```

When you use the /BATCH qualifier, a file is created in your current default directory that contains information about the batch job. This file has the same file name as the *input-file-spec* used on the command line with a default file type of LOG. When your batch job completes, this file is printed to the queue defined by the logical SYS$PRINT and the file is deleted.

Note that any process logical names you enter on the DOCUMENT command line must be defined in your LOGIN.COM file, otherwise VAX DOCUMENT will be unable to translate the logical name during batch processing and will issue an error message.

/CONDITION=condition-name

Specifies a condition keyword for a conditionalized SDML input file. Using this qualifier has the same effect as using the <SET_CONDITION> tag at the beginning of your input file. This qualifier accepts a *condition-name* argument that is a text string used to mark the condition being set.

The /CONDITION qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /CONDITION qualifier, the /CONDITION qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/CONTENTS
/NOCONTENTS

Specifies whether a table of contents file should be produced. The /NOCONTENTS qualifier is the default and specifies that no table of contents should be produced. When the /CONTENTS qualifier is specified, VAX DOCUMENT creates a table of contents file with a file name of *input-filename*_CONTENTS.

If you place the <CONTENTS_FILE> tag in your SDML file and also specify /DEVICE_CONVERTER on the command line, the current table of contents file is included at the corresponding point in the final printable output file (filename.LN03, filename.PS, and so on). If you do not place the <CONTENTS_FILE> tag in your SDML file, the table of contents is not incorporated into your final output file, but is placed in the separate file *input-filename*_CONTENTS and processed separately.

The file type of this table of contents output file depends upon the destination keyword and the processing qualifiers you have selected on the DOCUMENT command line. You can use the /KEEP qualifier to retain any intermediate table of contents files.

If you do not specify /CONTENTS on the DOCUMENT command line when you process a file that contains a <CONTENTS_FILE> tag, VAX DOCUMENT issues warning messages and the most recent version of the table of contents file is included. Note that this may result in an out-dated table of contents being included in your document. If there is no previous table of contents file to be included, the device converter issues an error message.

The /CONTENTS qualifier is valid only if text formatting is being done. If /NOTEXT_FORMATTER is specified with the /CONTENTS qualifier, the /CONTENTS qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

Note that if you process your document with the /NOTAG_TRANSLATOR and /CONTENTS qualifiers, and the table of contents is included using the <CONTENTS_FILE> tag, the table of contents is both incorporated into your document and placed in the file *filename_CONTENTS* and processed separately.

Note: The maximum length of a VMS file name is 39 characters. If you want to generate a contents file, your input file name must have no more than 30 characters, because appending _CONTENTS to it adds 9 more characters.

/DEVICE_CONVERTER[=(device-keyword [,device-keyword...])] /NODEVICE_CONVERTER

Specifies whether the device converter should be run.

The device converter reads and processes an intermediate device-specific file and converts it to a file suitable for output on the destination device. The output from the device converter has the same file name as the input file specified on the command line, and a file type based on the destination keyword specified on the command line.

The /DEVICE_CONVERTER qualifier optionally accepts the following keywords for special processing of your file:

Device Keyword	Description
HORIZONTAL_OFFSET=*number-of-points*	Specifies where the text is to be positioned, relative to the left edge of the paper. The default horizontal offset for the page is one inch (72 points). The *number-of-points* argument specifies the number of points the text page should be moved to the right from the left edge of the paper. This argument must be zero or a positive integer. The left edge of the paper is assumed to be zero.
VERTICAL_OFFSET=*number-of-points*	Specifies where the text is to be positioned, relative to the top edge of the paper. The default vertical offset for the page is one inch (72 points). The *number-of-points* argument specifies the number of points the text page should be moved toward the page bottom from the top edge of the page. This argument must be zero or a positive integer. The top edge of the paper is assumed to be zero.
STARTING_PAGE=*folio-spec*	Specifies the beginning page number of the first page in a range of pages to be printed. If no ending page is specified, the rest of the file is printed. The *folio-spec* value can be any valid page number that VAX DOCUMENT produces on the page[1].

[1]See the definition of a folio-spec in the description of the /DEVICE_CONVERTER qualifier.

Device Keyword	Description
ENDING_PAGE=*folio-spec*	Specifies the ending page number of the last page in a range of pages to be printed. The *folio-spec* value can be any valid page number that VAX DOCUMENT produces on the page[1]. If both the ENDING_PAGE and NUMBER_OF_PAGES keywords are used together on the same command line, VAX DOCUMENT issues an error message.
NUMBER_OF_PAGES=*maximum-pages*	Specifies the number of pages to print, when no ENDING_PAGE keyword is specified. If both the ENDING_PAGE and NUMBER_OF_PAGES keywords are used together on the same command line, VAX DOCUMENT issues an error message. The *maximum-pages* value must be an integer specifying the total number of pages to print. If this number is specified as a number larger than the number of pages in the document (for example, 99999), all pages of the document are printed.
FALLBACK	Specifies that the device converter should not process the input file using the multinational character set. Use this keyword only when you do not want to use the multinational character set when processing a file for a device that does not support the multinational character set; for example, certain 7-bit line printers and terminals.

[1]See the definition of a folio-spec in the description of the /DEVICE_CONVERTER qualifier.

Use the STARTING_PAGE and ENDING_PAGE keywords to specify the page numbers that are to be processed. Each of these keywords accepts a *folio-spec* argument. A folio-spec has the following syntax:

[{folio-prefix}{separator}] {page-number}

The following list describes the rules for each of the folio-spec syntax elements.

- *page-number*

  Specifies the page number portion of a folio-spec. In the folio-spec 11–3, "3" is the page-number. The page-number can be a Roman number, an Arabic number, or an asterisk (*). Roman numbers specify a page in the preface of a document, Arabic numbers specify pages outside the preface section, and the asterisk (*) specifies the first page of the document section specified by the folio-prefix.

- *folio-prefix*

  Specifies the numbers or letters that prefix the folio-spec. A folio-prefix can be any of the following:

  — Any single letter. Specifies an appendix in your document, for example the letter "B" in the folio-spec B–6.

  — Any single number. Specifies a chapter number, for example the number "13" in the folio-spec 13–1.

  — The keywords GLOSSARY or INDEX. Specifies a page in the glossary or index, for example "INDEX" in the folio-spec INDEX–6.

  — The keyword PART*n* where *n* is an integer of one or greater. Specifies a section begun using a <PART_PAGE> tag, for example "PART2" in the folio-spec PART2–7.

  - An asterisk (*). Specifies the first section of your document.

- *separator*

  Specifies the character that separates the folio-prefix from the page-number. The separator can be any single character that is not a space, a number, or a letter. In the folio-spec 11–3, "–" is the folio-prefix. A separator must be omitted if no folio-prefix is specified.

The following DOCUMENT command specifies that pages 11–3 through 11–8 of file MYREPORT.DVI_LN03 should be processed by the device convertor.

```
$ DOCUMENT/NOTAG/NOTEXT myreport REPORT LN03 -
_$ /DEVICE=(STARTING=11-3,ENDING=11-8)
```

/DIAGNOSTICS[=file-spec] /NODIAGNOSTICS

Causes the tag translator to write VAX Language-Sensitive Editor (LSE) diagnostics records to a file. LSE uses these records during its REVIEW phase to locate and describe translation errors. See Appendix B for more information on LSE.

If you omit the file specification, the output file has the same name as the input file, with a file type of DIA. The default qualifier is /NODIAGNOSTICS.

The /DIAGNOSTICS qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /DIAGNOSTICS qualifier, the /DIAGNOSTICS qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/ELEMENT=file-spec

Names the file specification of the book element that includes the input file specified on the DOCUMENT command line. If you use the /ELEMENT qualifier, the /PROFILE qualifier must also be used to specify the profile for the book that contains the book element. If you do not specify the file type of the element file, the default element file type is SDML.

The /ELEMENT qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /ELEMENT qualifier, the /ELEMENT qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/INCLUDE=file-spec

Specifies a VAX DOCUMENT file that you want to be included before the input file you specified on the command line. If you do not specify the file type of the file to be included, SDML is the default file type.

The /INCLUDE qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /INCLUDE qualifier, the /INCLUDE qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/INDEX[=(index-keyword [,index-keyword...])] /NOINDEX

Specifies whether an index file should be produced. The /NOINDEX qualifier is the default and specifies that no index should be produced. When the /INDEX qualifier is specified, VAX DOCUMENT creates an index file with a file name of *input-filename*_INDEX.

If you place the <INDEX_FILE> tag in your SDML file and also specify /DEVICE_CONVERTER on the command line, the current index file is included at the corresponding point in the final printable output file (filename.LN03, filename.PS, and so on). If you do not place the <INDEX_FILE> tag in your SDML file, the index is not incorporated into your final output file, but is placed in the separate file *input-filename_*INDEX.

The file type of this index output file depends upon the destination keyword and the processing qualifiers you have selected on the DOCUMENT command line. You can use the /KEEP qualifier to retain any intermediate index files.

If you do not specify /INDEX on the DOCUMENT command line when you process a file that contains an <INDEX_FILE> tag, VAX DOCUMENT issues a warning message and the most recent version of the index file is included. This may result in an out-dated index being included in your document. If no previous index file exists, the device converter issues an error message.

The /INDEX qualifier is valid only if text formatting is being done. If /NOTEXT_FORMATTER is specified with the /INDEX qualifier, the /INDEX qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

Note that if you process your document with the /NOTAG_TRANSLATOR and /INDEX qualifiers, and the index is included using the <INDEX_FILE> tag, the index is both incorporated into your document and placed in the file *filename_INDEX* and processed separately.

Note: The maximum length of a VMS file name is 39 characters. If you want to generate an index file, your input file name must have no more than 33 characters, because appending _INDEX to it adds 6 more characters.

The following list describes the optional indexing keywords that can be used with the /INDEX qualifier.

Index Keyword	Description
{ GUIDE_HEADINGS / NOGUIDE_HEADINGS }	Specifies whether alphabetic headings are created for each letter group in the index. (The entries beginning with "A" will have an A at the start of the group, and so on.) The GUIDE_HEADINGS keyword is the default. The NOGUIDE_HEADINGS keyword suppresses guide headings in the index output file.
SORT=([{ LETTER / WORD } [, { NONALPHA=AFTER / NONALPHA=BEFORE / NONALPHA=IGNORE }])	Specifies the sorting algorithm used to order entries in an index. • SORT=LETTER sorts the entries letter by letter and ignores spaces and hyphens. SORT=LETTER is the default. • SORT=WORD sorts the entries letter by letter and treats spaces and hyphens as significant. • SORT=NONALPHA positions entries with initial nonalphanumeric characters in the index based on the keyword supplied with the NONALPHA keyword. — The AFTER keyword causes entries with initial nonalphanumeric characters to be placed at the end of the index. — The BEFORE keyword causes entries with initial nonalphanumeric characters to be placed at the beginning of the index. — The IGNORE keyword causes entries with initial nonalphanumeric characters to be sorted by the first alphanumeric characters in the entry. The default is NONALPHA=IGNORE.
{ OVERRIDE_MASTER / NOOVERRIDE_MASTER }	Specifies the disposition of index entries in both single-document indexes and master indexes; NOOVERRIDE_MASTER is the default keyword for both /INDEX and /MASTER_INDEX. Use /INDEX=NOOVERRIDE_MASTER to create a single document index that contains only the index entries that are not marked with the MASTER keyword; this is the default. Use /INDEX=OVERRIDE_MASTER to create a single document index that contains both the index entries that are not marked with the MASTER keyword and the entries that are marked with the MASTER keyword. Use /MASTER_INDEX=NOOVERRIDE_MASTER to create a master index that contains only the index entries that are marked with the MASTER keyword; this is the default. Use /MASTER_INDEX=OVERRIDE_MASTER to create a master index that contains both the index entries that are not marked with the MASTER keyword and those entries that are marked with the MASTER keyword.

/KEEP[=(filetype-keyword [,filetype-keyword...])] /NOKEEP

Specifies whether intermediate files should be kept or deleted. The default qualifier /NOKEEP indicates that the intermediate files should be deleted by VAX DOCUMENT after processing.

The following keywords can be used to specify individual intermediate files that are to be kept:

File Type Keyword	File Contents
DVI	Specifies an intermediate output file from the text formatter. You may want to keep this file for reprocessing at a later date, or to selectively process and print only certain pages. The actual file type of this file is based on the destination keyword you specified on the DOCUMENT command line. These file types and the destination keywords that create them are listed under the description of the /TEXT_FORMATTER qualifier.
INX	Specifies an ASCII file that contains index entries in page-number order. This file can be used to create master indexes.
TEX	Specifies an input file for the text formatter. Note that a TEX file processed under a certain doctype design can produce errors if it is reprocessed using a different doctype.

/LIST[=file-spec] /NOLIST

Specifies whether a listing file is produced. The /NOLIST qualifier is the default and suppresses the generation of a listing file. If no file specification is provided as an argument to the /LIST qualifier, the qualifier causes a listing file to be produced with the file name of the input file specification and a default file type of LIS. If a file specification is provided as an argument, then that file specification is used as the output file.

The listing file contains the following information:

- All messages generated by the tag translator
- All messages generated by the text processor
- All messages generated by the device converter
- A brief summary section. This summary includes the following information:
  - The original command line
  - The time and day that processing began
  - The total CPU time used

/LOG /NOLOG

Specifies whether informational messages should be issued during processing. The /LOG qualifier is the default and specifies that informational messages should be issued. The /NOLOG qualifier suppresses informational messages.

The following example shows the typical informational messages displayed during tag translation (not specifying /NOLOG is the same as specifying the default qualifier /LOG):

```
$ DOCUMENT myarticle ARTICLE LN03

%DOC-I-IDENT, VAX DOCUMENT V1.1

[ T a g   T r a n s l a t i o n ]...
%TAG-I-DEFSLOADD, End of Loading of Tag Definitions
%TAG-I-ENDPASS_1, End of first pass over the input
[ T e x t   F o r m a t t i n g ]...
```

The following example shows how no messages are displayed when /NOLOG is specified. Note that only the DCL dollar ($) prompt is returned after the DOCUMENT command is finished.

```
$ DOCUMENT myarticle ARTICLE LN03 /NOLOG
$
```

/MAP[=file-spec]
/NOMAP

Specifies whether all the input files processed by VAX DOCUMENT should be listed in a file, which starts with the SDML input file and includes the tag table and any SDML files specified by the /INCLUDE or /SYMBOLS qualifier or the <ELEMENT> or <INCLUDE> tags. Files that are included by other files are indented under those files in this listing. The default file type of the file to be created is MAP_LIS. The default qualifier is /NOMAP.

The /MAP qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /MAP qualifier, the /MAP qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/MASTER_INDEX[=(index-keyword [,index-keyword...])]
/NOMASTER_INDEX

Causes indexing files to be collated into a master index file.

When the /MASTER_INDEX qualifier is specified, VAX DOCUMENT expects a master index data file as the *input-file-spec* parameter. This file lists the index output (INX) files generated from the individual books that are being referenced in the master index. The default file type of this input file is INX_LIST.

This qualifier accepts the same optional indexing keywords as the /INDEX qualifier; see the description of the /INDEX qualifier for a list of these keywords and their uses. See Chapter 7 for more information on creating a master index.

/OUTPUT=file-spec

Specifies a new name for the output file. If the /OUTPUT qualifier is not used, the output file is given the same file name as the input file, but with a file type based on the other qualifiers and destinations chosen.

The output file type is determined by the output device specified by the destination keyword. An example of a command line that uses the /OUTPUT qualifier follows:

```
$ DOC somefile LETTER LN03 /NOTEXT_FORMATTER /OUTPUT=anotherfile
```

This command line would create the file ANOTHERFILE.TEX because the processing was completed before the text formatter was run by the /NOTEXT_FORMATTER qualifier. If /NOTEXT_FORMATTER had been omitted from the preceding command line, the output file would have been ANOTHERFILE.LN03

The following table lists the output file types and the default destination keywords that produce them.

Default Destination Keyword	Default File Type
LINE	LINE
LN03	LN03
PS	PS
TERMINAL	TERM
MAIL	TXT

/PRINT[=(qualifier-keyword [,qualifier-keyword...])] /NOPRINT

Specifies whether the output file should be printed. The /PRINT qualifier builds and issues a DCL PRINT command. You can use any of the DCL PRINT command qualifiers with VAX DOCUMENT by passing these qualifiers as keywords to the /PRINT qualifier.

For example, if you want to print two copies of a VAX DOCUMENT file with no flag page, you can use the PRINT command as follows:

```
$ PRINT somefile.line /NOFLAG/COPIES=2
```

You can specify the same options using the following /PRINT qualifier on the DOCUMENT command line.

```
$ DOC/NOTAG/NOTEXT/NODEV somefile -
_$/PRINT=(NOFLAG,COPIES=2) LETTER LINE
```

When you use the /PRINT qualifier, you do not need to specify an output device; VAX DOCUMENT determines the appropriate output device from the destination keyword specified on the DOCUMENT command line.

Default print queues for each of the destination keywords available at your site are established when VAX DOCUMENT is installed. These queues include all DCL PRINT command qualifiers needed for correct printing on that output device.

If you choose to specify a print queue other than the default, you must also specify the correct qualifier-keywords needed for that queue. These keywords vary depending on the type of file to be printed (indicated by the file extension) and the type of output device.

The following table shows the keywords to use for each file and output device combination.

Output File Type	Print Device	DOCUMENT /PRINT Qualifier-Keywords
LINE	Line Printer LN03 Laser Printer LN03 PLUS Laser Printer	/PRINT=(QUEUE=queuename)
	LN03R SCRIPTPRINTER PRINTSERVER 40	/PRINT=(QUEUE=queuename,PARAM=DATA_TYPE=ANSI)
LN03	LN03 Laser Printer LN03 PLUS Laser Printer	/PRINT=(QUEUE=queuename,NOFEED,PASSALL)
	LN03R SCRIPTPRINTER PRINTSERVER 40	/PRINT=(QUEUE=queuename,PARAM=DATA_TYPE=ANSI)
PS	LN03R SCRIPTPRINTER PRINTSERVER 40	/PRINT=(QUEUE=queuename,PARAM=DATA_TYPE=POST)

If you wish to print your file using the DCL PRINT command, the following table shows the qualifiers you must use for each supported device.

Output File Type	Print Device	DCL PRINT Command Qualifiers
LINE	Line Printer LN03 Laser Printer LN03 PLUS Laser Printer	$ PRINT filename.LINE _$ /QUEUE=queuename
	LN03R SCRIPTPRINTER PRINTSERVER 40	$ PRINT/PARAM=(DATA_TYPE=ANSI) filename.LINE _$ /QUEUE=queuename
LN03	LN03 Laser Printer LN03 PLUS Laser Printer	$ PRINT/NOFEED/PASSALL filename.LN03 _$ /QUEUE=queuename
	LN03R SCRIPTPRINTER PRINTSERVER 40	$ PRINT/PARAM=(DATA_TYPE=ANSI) filename.LN03 _$ /QUEUE=queuename
PS	LN03R SCRIPTPRINTER PRINTSERVER 40	$ PRINT/PARAM=(DATA_TYPE=POST) filename.PS _$ /QUEUE=queuename

When you specify /DEVICE_CONVERTER on the DOCUMENT command line, /PRINT is the default printing qualifier when the destination keywords are LN03, PS, or LINE. The /NOPRINT qualifier is the default when the destination keywords are TERMINAL or MAIL.

Note that if any serious errors are encountered during processing, the file is not sent to the print device.

/PROFILE=file-spec

Specifies that the input file is a file referenced by the <ELEMENT> tag in a bookbuild profile and that a cross-reference data file created by a VAX DOCUMENT bookbuild should be used to resolve any cross references in the input file. The cross-reference data file is specified as an argument to the /PROFILE qualifier and has a file type of XREF.

The /PROFILE qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /PROFILE qualifier, the /PROFILE qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/SYMBOLS=file-spec /NOSYMBOLS

Controls whether a file of symbol definitions is to be read automatically during tag translation. The default qualifier is /NOSYMBOLS. The SDML input file accessed through the /SYMBOLS qualifier should contain only symbol definitions created by the <DEFINE_SYMBOL> and the <DEFINE_BOOK_NAME> tags. The default file type of this file is SDML.

The /SYMBOLS qualifier is valid only if tag translation is being done. If /NOTAG_TRANSLATOR is specified with the /SYMBOLS qualifier, the /SYMBOLS qualifier is ignored, and VAX DOCUMENT issues an informational message stating that you have specified conflicting qualifiers.

/TAG_TRANSLATOR /NOTAG_TRANSLATOR

Controls whether the tag translator is run. If this qualifier is not specified, the default qualifier is /TAG_TRANSLATOR.

If the *input-file-spec* parameter specified on the command line does not include a file type, the default file type is SDML. By default, the tag translator produces an output file with the file name of the input file and a file type of TEX. This file can then be used as an input file to the text formatter.

/TEXT_FORMATTER /NOTEXT_FORMATTER

Controls whether the text formatting program is run. If /NOTEXT_FORMATTER is specified, VAX DOCUMENT processing stops after the tag translator completes. /TEXT_FORMATTER is the default.

If /NOTAG_TRANSLATOR is specified and the input file specified on the command line does not include a file type, the default file type is TEX. The output file from the text formatter can have one of several file types based on the destination parameter entered on the command line. These default output file types and their related destination keywords are given in the following list.

Destination Keyword	Intermediate Output File Type	Final Output File Type
LINE	DVI_LINE	LINE
LN03	DVI_LN03	LN03
MAIL	DVI_LINE	TXT
PS	DVI_PS	PS
TERMINAL	DVI_LINE	TERM

The output file from the text formatter can then be used as an input file to the device converter.

EXAMPLE

When you process your input file using the DOCUMENT command, you are actually running your file through several processors. The following example shows a typical use of the VAX DOCUMENT command:

```
$ DOCUMENT myreport.SDML REPORT LN03 /CONTENTS

%DOC-I-IDENT, VAX DOCUMENT V1.1

[ T a g   T r a n s l a t i o n ]...
%TAG-I-DEFSLOADD, End of Loading of Tag Definitions
%TAG-I-ENDPASS_1, End of first pass over the input
[T e x t   F o r m a t t i n g ]...
%TEX-I-PAGESOUT, 17 pages written.
-TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYREPORT.DVI_LN03'
[ C o n t e n t s   G e n e r a t i o n ]...
[ T e x t   F o r m a t t i n g   C o n t e n t s ]...
%TEX-I-PAGESOUT, 1 page written.
-TEX-I-OUTFILENAME, 'DUA1:[DOCFILES]MYREPORT_CONTENTS.DVI_LN03'
[ D e v i c e   C o n v e r s i o n ]...
%DVC-I-PAGESOUT, 18 pages written to file:
   DUA1:[DOCFILES]MYREPORT.LN03
[ C o n t e n t s   D e v i c e   C o n v e r s i o n ]...
%DVC-I-PAGESOUT, 1 page written to file:

   DUA1:[DOCFILES]MYREPORT_CONTENTS.LN03
[ P r i n t i n g   F i l e ]...
Job MYREPORT (queue SYS$LN03, entry 833) started on SYS$LN03

[ P r i n t i n g   C o n t e n t s]...

Job MYREPORT_CONTENTS (queue SYS$LN03, entry 834) started on SYS$LN03

$
```

B Using LSE with VAX DOCUMENT

This appendix provides an overview of an optional productivity tool, the VAX Language-Sensitive Editor. This tool is not included with the VAX DOCUMENT software; it must be purchased separately. Using LSE can increase your productivity as a VAX DOCUMENT user. For information on how to purchase this tool, contact your DIGITAL sales representative.

B.1 Using LSE with VAX DOCUMENT

The VAX Language-Sensitive Editor (LSE) is a powerful and flexible text editor designed specifically for software development. LSE has important features that help you produce syntactically correct SDML markup in VAX DOCUMENT.

To invoke LSE, issue the LSEDIT command followed by a file name with a SDML file type at the DCL prompt. For example:

```
$ LSEDIT USER.SDML
```

The following sections describe some of the key features of LSE. Section B.1.1 discusses how to enter source code using LSE and Section B.1.2 describes LSE's compiler interface features. Section B.1.3 gives examples of how to generate VAX DOCUMENT source code with LSE.

For more details on advanced features of LSE, see the *Guide to VAX Language-Sensitive Editor and VAX Source Code Analyzer.*

B.1.1 Entering Source Code Using Tokens and Placeholders

LSE simplifies the tasks of developing and maintaining documents. LSE provides the functions of a traditional text editor, plus additional powerful features: language-specific placeholders and tokens, aliases, compile and review features.

Placeholders are markers in the source code that indicate locations where you can provide text. Placeholders help you to supply the appropriate syntax in a given context. Generally, you do not need to type placeholders; rather, they are inserted for you by LSE.

Placeholders are either optional or required. Required placeholders, which are delimited by braces ({}), represent places in the source code where you must provide text. Optional placeholders, which are delimited by brackets ([]), represent places in the source code where you can either provide additional constructs or delete the placeholder.

There are three types of LSE placeholders.

Type of Placeholder	Description
Terminal	Provides text that describes valid replacements for the placeholder
Nonterminal	Expands into additional language constructs
Menu	Provides a list of options corresponding to the placeholder

You can move forward or backward from placeholder to placeholder. In addition, you can delete or expand placeholders as needed. Section B.1.3 shows examples of expanding placeholders.

Tokens typically represent each tag in VAX DOCUMENT. When expanded, tokens provide the complete syntax of a tag. You can type tokens directly into the buffer. Generally, you use tokens when you want to add a tag and there are no placeholders in an existing program. For example, typing COM and issuing the EXPAND command causes the tags for a comment to appear on your screen. You can also use tokens to by-pass long menus in cases where expanding a placeholder, such as [element-or-template], would result in a lengthy menu.

You can use tokens to insert text when editing an existing file by typing the name for a function or keyword and issuing the EXPAND command.

LSE commands allow you to manipulate tokens and placeholders. These commands and their default key bindings are as follows:

Table B–1 LSEDIT Commands

Command	Key Binding	Function
EXPAND	CTRL/E	Expands a placeholder
UNEXPAND	PF1-CTRL/E	Reverses the effect of the most recent placeholder expansion
GOTO PLACEHOLDER/FORWARD	CTRL/N	Moves the cursor to the next placeholder
GOTO PLACEHOLDER/REVERSE	CTRL/P	Moves the cursor to the previous placeholder
ERASE PLACEHOLDER/FORWARD	CTRL/K	Erases a placeholder
UNERASE PLACEHOLDER	PF1-CTRL/K	Restores the most recently erased placeholder
None	Down arrow	Moves the indicator down through a menu
None	Up arrow	Moves the indicator up through a menu
None	{ ENTER / RETURN }	Selects a menu option

You can display a list of all defined tokens and placeholders, or a particular token or placeholder, with the LSE commands— SHOW TOKEN and SHOW PLACEHOLDER.

To copy the listed information into a separate file, first issue the appropriate SHOW command to put the list into the $SHOW buffer. Then issue the following commands:

```
LSE> GOTO BUFFER $SHOW
LSE> WRITE filename
```

To obtain a hard copy of the list, use the PRINT command at DCL level to print the file you created.

B.1.2 Compiling Source Code

To compile your code and review compilation errors without leaving the editing session, you can use the LSE commands COMPILE and REVIEW. The COMPILE command issues a DCL command in a subprocess to invoke the VAX DOCUMENT tag translator. The compiler then generates a file of compile-time diagnostic information that LSE can use to review compilation errors. The diagnostic information is generated with the /DIAGNOSTICS qualifier that LSE appends to the compilation command.

For example, to issue the COMPILE command while in the buffer USER.SDML, you would use the following command:

```
COMPILE $ doctype destination
```

The *doctype* and *destination* are any valid VAX DOCUMENT keywords. When the COMPILE command is issued, the following DCL command executes:

```
$ DOCUMENT USER.SDML/DIAGNOSTICS=USER.DIA doctype destination
```

LSE supports all of VAX DOCUMENT's command qualifiers.

The REVIEW command displays any diagnostic messages that result from a compilation. LSE displays the compilation errors in one window and the corresponding source code in a second window so that you can review your errors while examining the associated source code. This capability eliminates tedious steps in the error correction process and helps ensure that all the errors are fixed before you recompile your program.

LSE provides several commands to help you review errors and examine your source code. These commands, and their default key bindings where applicable, are as follows:

Command	Key Binding	Function
COMPILE	None	Compiles the contents of the source buffer. You can issue this command with the /REVIEW qualifier to put LSE in REVIEW mode immediately after the compilation.
REVIEW	None	Puts LSE into REVIEW mode and displays any errors resulting from the last compilation.
END REVIEW	None	Removes the buffer $REVIEW from the screen; returns the cursor to a single window containing the source buffer.
GOTO SOURCE	CTRL/G	Moves the cursor to the source buffer that contains the error.

Command	Key Binding	Function
NEXT STEP	CTRL/F	Moves the cursor to the next error in the buffer $REVIEW.
PREVIOUS STEP	CTRL/B	Moves the cursor to the previous error in the buffer $REVIEW.
	{ Down arrow Up arrow }	Moves the cursor within a buffer.

B.1.3 Examples

The following sections show examples of using some common tokens and placeholders to write VAX DOCUMENT code. The examples are expanded to show the formats and guidelines LSE provides; however, not all of the examples are fully expanded.

The examples show expansions of the following VAX DOCUMENT features:

- Tags as tokens: <LIST> and <TABLE>
- Initial string: [element-or-template]
- Sample template: Data Item Description (DID) for U.S. Department of Defense military specification

Instructions and explanations precede each example, and an arrow (←) indicates the line in the code where an action has occurred.

See Table B–1 for the commands that manipulate tokens and placeholders.

Remember that braces ({}) enclose required placeholders; brackets ([]) enclose optional placeholders. Note that when you erase an optional placeholder, LSE also deletes any associated text before and after that placeholder.

When the editor is used to create a new VAX DOCUMENT file, the initial string, {element-or-template}, will appear at the top of the screen. Expansion of the initial string will produce the following

```
-> [profile]
   [front_matter]
   [chapter]
   [part]
   [appendix]
   [glossary]
   [tag-groups]
   [article-tags]
   [letter-tags]
   [report-tags]
   [overheads-tags]
   [milspec-templates]
   [software-documentation-templates]
```

A convention in VAX DOCUMENT is to place a single element, such as a chapter or appendix, or a single template, such as a milspec template or a software documentation template, into a single file.

B.1.3.1 Lists

Select the token LIST by typing LIS and expanding the token.

From the menu of possible list types, select the option Alphabetical-list. The following then appears on the screen in the editing buffer:

```
<LIST>(ALPHABETIC\[start-no]\[case])
{list-element}...
<ENDLIST>
```

Erase the placeholder [start-number] and expand the placeholder [case]. Select UPPERCASE.

```
<LIST>(ALPHABETIC\UPPERCASE)
{list-element}...
<ENDLIST>
```

Move to the next placeholder and expand to get an individual list element tag.

```
<LIST>(ALPHABETIC\UPPERCASE)
<LE>{text}
[list-element]...
<ENDLIST>
```

Type over the {text} placeholder to insert the text for the first list item.

B.1.3.2 Tables

Enter all or part of the token TABLE and expand.

```
<TABLE>({caption}\[symbol])
[<TABLE_ATTRIBUTES>]
<TABLE_SETUP>({number}[\col-width]...)
<TABLE_HEADS>({heading}[\heading]...)
{rows}...
<ENDTABLE>
```

The <TABLE> tag takes two arguments: the caption to the table and a symbol to use for cross-references. With the cursor positioned on {caption}, enter caption text. With the cursor positioned on [symbol], enter symbol text.

```
-> <TABLE>(Caption for the Table\newsymbol_tab)
   [<TABLE_ATTRIBUTES>]
   <TABLE_SETUP>({number}[\col-width]...)
   <TABLE_HEADS>({heading}[\heading]...)
   {rows}...
   <ENDTABLE>
```

Next, expand the placeholder <TABLE_ATTRIBUTES> and choose a set of attributes from the menu.

```
   <TABLE>(Caption for the Table\newsymbol_tab)
-> <TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
   <TABLE_SETUP>({number}[\col-width]...)
   <TABLE_HEADS>({heading}[\heading]...)
   {rows}...
   <ENDTABLE>
```

Next, set up the table for the number and width of columns desired. Enter a number indicating the number of columns (2 through 9). Then expand the [\col-width] placeholder to enter a number indicating the width of the first table column.

```
    <TABLE>(Caption for the Table\newsymbol_tab)
    <TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
->  <TABLE_SETUP>(3\{number}[\col-width]...)
    <TABLE_HEADS>({heading}[\heading]...)
    {rows}...
    <ENDTABLE>
```

Notice that the expansion of [\col-width] gives you the backslash needed to separate arguments and the {number} placeholder. If you enter a number into [\col-width] without expanding it first, you would not get the backslash. Enter column widths for each column in the table except the last column. Delete the last [\col-width] placeholder.

```
    <TABLE>(Caption for the Table\newsymbol_tab)
    <TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
->  <TABLE_SETUP>(3\15\15 )
    <TABLE_HEADS>({heading}[\heading]...)
    {rows}...
```

The {heading} placeholders work very similarly to the [\col-width] placeholder:

```
    <TABLE>(Caption for the Table\newsymbol_tab)
    <TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
    <TABLE_SETUP>(3\15\15 )
->  <TABLE_HEADS>(First heading\Second Heading\{heading}[\heading]...)
    {rows}...
    <ENDTABLE>
```

Enter a heading for each column and delete the last [\heading] placeholder.

Expand the {rows} placeholder to get tags for the body of the table.

```
    <TABLE>(Caption for the Table\newsymbol_tab)
    <TABLE_ATTRIBUTES>(WIDE\MULTIPAGE)
    <TABLE_SETUP>(3\15\15 )
    <TABLE_HEADS>(First heading\Second Heading\{heading}[\heading]...)
->  <TABLE_ROW>({item}[\item]...)
    [rows]...
```

Enter an item for each column and keep expanding the [rows] placeholder until the table body is complete. Delete the last [rows] placeholder and the table is ready for processing.

B.1.3.3 Profile

To place the tags for a document's profile into the editing buffer, type PRO and expand the token.

```
    <PROFILE>

          {element}...

    <ENDPROFILE>
```

Expand the placeholder {element} to get the <ELEMENT> tag and the optional <INCLUDES_FILE> tag.

```
<PROFILE>
->      <ELEMENT>({element-name})
           [<INCLUDES_FILE>]
        [element]...
<ENDPROFILE>
```

Expand or delete the placeholder <INCLUDES_FILE> and expand the placeholder [element] as many times as needed.

B.1.3.4 Sample Template

There are two methods of accessing the Data Item Description (DIDs) templates for DOD-STD-2167-related military specifications:

- Select [milspec-templates] from the initial placeholder
- Enter part of the token "DID-800..." and expand to get a list of the available data item descriptions

Select one of the DIDs from the list.

```
-> DI-MCCR-80012:  Software Top Level Design Document
   DI-MCCR-80025:  Software Requirements Specification
   DI-MCCR-80026:  Interface Requirements Specification
   DI-MCCR-80027:  Interface Design Document
   DI-MCCR-80028:  Data Base Design Document
   DI-MCCR-80031:  Software Detailed Design Document
```

Select one of the DIDs, for example, DI-MCCR-80012. The menu that appears is a list of templates that make up the individual files in the Software Top Level Design Document.

For example, select DI-MCCR-80012__cover_page:

```
   DI-MCCR-80012__profile
   DI-MCCR-80012__symbols
-> DI-MCCR-80012__cover_page
   DI-MCCR-80012__Scope
   DI-MCCR-80012__Referenced_Documents
   DI-MCCR-80012__Requirements
   DI-MCCR-80012__Qualification_Requirements
   DI-MCCR-80012__Preparation_for_Delivery
   DI-MCCR-80012__Notes
   DI-MCCR-80012__Appendix
```

Each of the menu items displayed for DI-MCCR-80012 contains a template for a section of the Software Top Level Design Document and should be placed in a separate file.

```
<comment>(begin front matter component)
<front_matter>
<title_page>
<specification_info>(<reference>(control_number)\\<DATE>\)
<spec_title>(<reference>(DID80012_title)\For the\{text}\
Contract No. <reference>(contract_number)\
CDRL Sequence No. <reference>(CDRL_number)\<DATE>)

<subtitle>(Prepared for:\{text}\ \
Prepared by:\{text}\{text})
```

```
<endtitle_page>

<contents_file>

<endfront_matter>
<comment>(end front matter component)
```

The tags in the cover page template mark standard title page text. Fill in the variable information by typing over the {text} placeholders.

B.1.4 VAX DOCUMENT Tokens and Placeholders

To see all of the defined tokens provided by VAX DOCUMENT, enter the following Editor command:

```
LSE> SHOW TOKEN
```

To see all of the defined placeholders provided by the VAX DOCUMENT, enter the following Editor command:

```
LSE> SHOW PLACEHOLDER
```

To print a copy of either of these lists, you must first enter the appropriate SHOW command. This enters the list into the $SHOW buffer. Then enter the following commands:

```
LSE> GOTO BUFFER $SHOW
LSE> WRITE filename
```

At DCL level, you can use the PRINT command to obtain a hard copy of the list.

You may also specify a token name or placeholder name after the SHOW TOKEN or SHOW PLACEHOLDER command to obtain information about a particular token or placeholder.

C Messages

A message has the following format:

```
% facility--severity--identification,  text
```

Messages in VAX DOCUMENT can come from the following facilities:

Facility code	Facility
DOC	The DOCUMENT command line
TAG	The tag translator
TEX	The text processor
DVC	The device converter
INX	The index facility

The messages from these sources are given in the following sections.

To locate a message, use the facility code to locate the appropriate section and then look for the identification text in that section. Messages are alphabetical within sections.

For example, consider the following message:

```
%TAG-W-ISTHISTAG, at text on line 245 in file
   WRT_:[YOURNAME.BOOK]INTRO.GNC;
   Ignoring <list. Is this a tag without a closing angle bracket?
```

The facility code in this error is TAG, so you look in the section for tag translator message. The identification text is ISTHISTAG, so you locate that within the section.

C.1 DOCUMENT Command Messages

ACTION_EXCL , *qualifier* is not available for the chosen destination

Warning: Indicates that the default qualifier has been excluded for the desired destination and will therefore be ignored.

User Action: None.

AMB_DESIGN , Ambiguous design keyword *design-keyword*

Fatal: An ambiguous design keyword was specified.

User Action: Change the specified design keyword to be a unique design keyword.

AMB_DEST , Ambiguous destination keyword *destination-keyword*

Fatal: An ambiguous destination keyword was specified.

User Action: Change the specified destination keyword to be a unique keyword.

AMB_PAPER_SIZE , Ambiguous paper size value

Error: The specified paper size value is ambiguous.

User Action: Specify a unique paper value and reexecute the DOCUMENT command.

CANT_GET_VALUE , Cannot get value of /*qualifier* qualifier

Fatal: A required value associated with the qualifier was not provided.

User Action: Reexecute the DOCUMENT command and specify a value for the qualifier.

CANT_OPEN , Error detected opening file *file-spec*

Error: An error has occurred while opening the specified file.

User Action: Fix problem if you can or else refer problem to your system manager. After fixing the problem, reexecute the DOCUMENT command.

CANT_USE_SHR , Cannot use *file-spec*

Fatal: A shareable image file for text processing could not be found or used.

User Action: Consult system manager.

DCL_ERROR , An error was detected while parsing *DCL-command*\\

Error: An error was detected by the DCL parser.

User Action: Correct error on the specified line and reexecute the DOCUMENT command.

DESIGN_ERROR , Bad design entry in *design-file-directory* DOC$DESIGNS data file

Error: An error was found in a design data file, erroneous design entries are ignored.

User Action: Correct error and reexecute the DOCUMENT command.

DESIGN_SYNTAX , Ignoring *design-keyword* design entry

Error: A syntax error is detected in the design data file.

User Action: Correct error and reexecute the DOCUMENT command.

DEST_ERROR , Bad destination entry in *destination-file-directory* DOC$DESTINATIONS data file

Error: An error was found in a destination data file. Any involved entries ignored.

User Action: Correct error and reexecute the DOCUMENT command.

DEST_SYNTAX , Ignoring *destination-keyword* destination data entry

Error: An error has been detected in a destination data file.

User Action: Correct errors in the destination data file and reexecute the DOCUMENT command.

DO_CONTENTS , [Contents Generation]...

Informational: Indicates that a contents file is being created.

User Action: None.

DO_DEVICE , [Device Conversion]...

Informational: Indicates that the document input is being processed into a printable form.

User Action: None.

DO_DVC_CONTENTS , [Contents Device Conversion]...

Informational: Indicates that the document contents is being processed into a printable form.

User Action: None.

DO_DVC_INDEX , [Index Device Conversion]...

Informational: Indicates that the document index is being processed into a printable form.

User Action: None.

DO_INDEX , [Index Generation]...

Informational: Indicates that an index file is being created.

User Action: None.

DO_MASTER_INDEX, [Master Index Generation]...

Informational: Indicates that a master index file is being created.

User Action: None.

DO_PRINT, [Printing File]...

Informational: Indicates that your output file is being sent to the designated print queue.

User Action: None.

DO_TAG, [Tag Translation]...

Informational: Indicates that the tag translator is processing your SDML input file.

User Action: None.

DO_TEXT, [Text Formatting]...

Informational: Indicates that the text formatter is processing.

User Action: None.

DO_TEXT_CONTENTS, [Text Formatting Contents]...

Informational: Indicates that the text formatter is processing the contents file.

User Action: None.

DO_TEXT_INDEX, [Text Formatting Index]...

Informational: Indicates that the text formatter is processing the index file.

User Action: None.

ERROR_CONVERTER, Errors found by the device converter

Error: The device converter has ended in error, processing cannot continue.

User Action: Correct errors found by the device converter and reexecute the DOCUMENT command.

ERROR_FORMATTER, Errors found by the text formatter

Error: The text formatter has ended in error, processing cannot continue.

User Action: Correct errors found by the text formatter and reexecute the DOCUMENT command.

ERROR_IN_DESIGN, Internal design date file error

Fatal: An unspecified error in the design data file was found.

User Action: Refer problem to system manager. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ERROR_IN_DEST , Internal destination data file error

Fatal: An unspecified error in the destination data file was found.

User Action: Refer problem to system manager. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ERROR_TAG , Errors found by the tag translator

Error: The tag translator has ended in error, processing cannot continue.

User Action: Correct errors found by the tag translator and reexecute the DOCUMENT command.

ERROR_WRITE , Error writing to file *file-spec*

Fatal: An RMS error was reported when DOCUMENT tried to read the specified file.

User Action: Correct RMS error and reexecute DOCUMENT command.

ERR_CRE8_LIST , Listing file *file-spec* cannot be created

Fatal: The specified listing file could not be created.

User Action: Correct error in the listing file specification.

FILLM_TOO_LOW , Process open file limit too low, required value is *minimum-value*

Fatal: The FILLM of the current process in not large enough to run the DOCUMENT command.

User Action: Contact system manager to increase quota.

IDENT , VAX Document *version number*

Informational: DOCUMENT version number identification message

User Action: None.

IGNOR_DESIGN , Ignoring optional *logical-name* design data file

Informational: Optional DOC$LOCAL_FORMATS or DOC$STANDARD_FORMATS design file cannot be opened. Any designs which are in that file will be unavailable.

User Action: Correct reported problem or contact system manager, then reexecute the DOCUMENT command.

INVALID_DESIGN , *design-keyword* is not a valid design

Fatal: An unrecognized design parameter was specified on the DOCUMENT command line.

User Action: Reexecute the DOCUMENT command with a valid design parameter.

INVALID_DEST , *destination-keyword* is not a valid destination

Fatal: An unrecognized destination keyword was specified on the DOCUMENT command line.

User Action: Reexecute the DOCUMENT command with a valid destination keyword.

INVALID_DVI_ARG , *qualifier-arg* is not a valid /DEVICE_CONVERTER argument

Fatal: The argument value specified to /DEVICE_CONVERTER is unrecognized.

User Action: Reexecute DOCUMENT command with a valid qualifier argument.

INVALID_ENTRY , *qualifier-entry* is not a valid /*qualifier* entry

Fatal: An invalid qualifier entry was specified on the DOCUMENT command.

User Action: Reexecute the DOCUMENT command with a valid qualifier. Refer to Appendix A in this manual for information on valid DOCUMENT command qualifiers.

INVALID_INPUT , Cannot use *file-spec* as an input file

Fatal: The specified input file is invalid.

User Action: Reexecute the DOCUMENT command with a valid input file specification.

INVALID_VALUE , Cannot use *file-spec* as a file for /*qualifier*

Fatal: The specified input file associated with the qualifier cannot be opened.

User Action: Reexecute DOCUMENT command with a valid input file specification.

NEED_VALUE , Destination entry parameter command needs /VALUE or a keyword

Error: Syntax error found when parsing the destination data file.

User Action: Refer problem to system manager. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NO_CONTENTS , No contents; /CONTENTS qualifier unnecessary

Warning: No contents information was available to generate a contents file; no contents file will be produced.

User Action: Do not use the /CONTENTS qualifier when processing this input file. An SDML file must contain at least one <CHAPTER> , <APPENDIX> , or <HEADx> tag to produce a table of contents file.

NO_INDEX , No index; /INDEX qualifier unnecessary

Warning: No index information was available to generate an index file; no index file will be produced.

User Action: Do not use the /INDEX qualifier when processing this input file. An SDML file must contain at least one <X> or <Y> tag to produce an index file.

OPEN_DESIGN , Error opening design data file in DOC$STANDARD_FORMATS

Fatal: An error occurred when trying to open the DOC$STANDARD_FORMATS:DOC$DESIGNS.DAT data file.

User Action: Check for existence of DOC$STANDARD_FORMATS:DOC$DESIGNS.DAT data file; if it exists, then check the file protection on that file.

OPEN_DEST , Error opening DOC$STANDARD_FORMATS:DOC$DESTINATIONS data file

Fatal: There was an RMS error when trying to open the destination data file.

User Action: Check for existence of DOC$STANDARD_FORMATS:DOC$DESTINATIONS.DAT data file; if it exists, then check the file protection on that file.

PGFLQ_TOO_LOW , Process paging file quota too low, required value is *minimum-value*

Fatal: The PGFLQUOTA of the current process is not large enough to run the DOCUMENT command.

User Action: Contact system manager to increase quota.

PRINT_CONTENTS , [Printing Contents]...

Informational: Indicates that your contents file is being sent to the designated print queue.

User Action: None.

PRINT_INDEX , [Printing Index]...

Informational: Indicates that your index file is being sent to the designated print queue.

User Action: None.

THESE_DESIGNS , Choose one of these designs:

Informational: Inquires for choice of document design types, this is required input to the DOCUMENT command.

User Action: At the prompt, type the desired design from the choices offered.

THESE_DESTS , Choose one of these destinations:

Informational: Inquires for choice of document destinations, this is required input to the DOCUMENT command.

User Action: At the prompt, type the desired destination from the choices offered.

TYPE_INPUT , input file

Informational: Inquires for the input file.

User Action: At the prompt, type the name of the input file.

UNR_DESIGN , Unrecognized design - please check your spelling

Fatal: An unrecognized design parameter has been specified to the DOCUMENT command.

User Action: Reexecute the DOCUMENT command with a valid design parameter.

UNR_DEST , Unrecognized destination - please check your spelling

Fatal: An unrecognized destination parameter has been specified to the DOCUMENT command.

User Action: Reexecute the DOCUMENT command with a valid destination parameter.

C.2 Tag Translator Messages

ABORTFORE, at tag or text on line *n* of file *file-spec*
There have been more than *number* error messages of severity level E

Fatal: The tag translator did not produce an output file because it detected too many errors in the SDML source file with a severity level of Error. SDML defines the minimum number of error messages of severity E as 0, although this default value can be modified in local usage.

User Action: Examine the output from the preceding messages to determine the errors you need to correct.

ABORTFORW, at tag or text on line *n* of file *file-spec*
There have been more than *number* error messages of severity level W

Fatal: The tag translator did not produce an output file because it detected too many errors in the SDML source file with a severity level of Warning. SDML defines the minimum number of error messages of severity W as 30, although this default value can be modified in local usage.

User Action: Examine the output from the preceding messages to determine the errors you need to correct.

ARGBADOPR, at tag or text on line *n* of file *file-spec*
Argument 2 to tag <COUNTER> has an illegal arithmetic operator

Warning: The tag translator has found an illegal operator. The second argument to the <COUNTER> tag consists of pairs of operators and operands, and an optional trailing comment. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the operators and the comment character are correct.

ARGINCMPL, at tag or text on line *n* of file *file-spec*
Argument 2 to tag <COUNTER> is incomplete.

Warning: The tag translator has not found at least one operator and one operand. The second argument to the <COUNTER> tag requires at least one operator and one operand. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the second argument begins with an operator and an operand.

ARGINVALD, at tag or text on line *n* of file *file-spec*
Argument *number* to tag <*tagname*> is an invalid argument.

Warning: The argument in the specified position is not valid for the specified tag.

User Action: Verify the tag's argument and correct it.

ARGMENTIS, The argument is *string*

Informational: Follows a message that indicates that an argument is invalid, for example, when a numeric argument is required, but a character string argument is specified.

User Action: Use the information to correct your source file.

ARGMISSNG, at tag or text on line *n* of file
file-spec
Argument *number* to tag *<tagname>* is missing

Warning: The tag translator found no text for the indicated argument, but the argument is required for the indicated tag.

User Action: Supply a value for the indicated argument.

ARGNOTCHR, at tag or text on line *n* of file
file-spec
Argument to tag <CHR> is not in the range 32 to 126

Warning: The <CHR> built-in tag requires a numeric argument whose value is in the range of 32 to 126. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Correct the reference to the <CHR> built-in tag so that it specifies a value that is in the correct numeric range.

ARGNOTKEY, at tag or text on line *n* of file
file-spec
Argument *number* to tag *<tagname>* is not a keyword

Warning: The argument in the indicated position is not a valid keyword argument for the indicated tag. This message is generally followed by a message that shows the keyword that was specified.

User Action: Using the information in this message and the accompanying message, determine the keyword in error. Check the *VAX DOCUMENT User Manual, Volume 1* to determine the valid arguments for the tag in question and correct your source file.

ARGNOTLET, at tag or text on line *n* of file
file-spec
Argument to tag <SET_APPENDIX_LETTER> is not a letter

Warning: The <SET_APPENDIX_LETTER> tag requires a letter (A to Z).

User Action: Correct the source file to supply a letter as the first character of the argument.

ARGNOTMSG, at tag or text on line *n* of file
file-spec
Argument *number* to tag <*tagname*> is not a valid message code

Warning: The indicated argument to the <TAG_DIAGNOSTIC> built-in tag is not a recognized message code. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify the spelling of the message code and correct your source file.

ARGNOTNAM, at tag or text on line *n* of file
file-spec
Argument *number* to tag <*tagname*> is not a valid name

Warning: The indicated argument does not obey the rules for formation of a name. A name must have only letters, digits and the underscore character. A name must not begin with two underscores. Spaces, tabs, and carriage returns are not allowed.

User Action: Correct the argument so that the name is valid.

ARGNOTNUM, at tag or text on line *n* of file
file-spec
Argument *number* to tag <*tagname*> is not a number

Warning: A non-numeric argument was specified to a tag that requires a numeric argument.

User Action: Verify that the argument is in the correct position in the tag's argument list. Supply a numeric argument to the tag.

ARGNOTPOS, at tag or text on line *n* of file
file-spec
Argument to tag <SET_CHAPTER_NUMBER> is not a positive integer

Warning: The <SET_CHAPTER_NUMBER> tag requires a positive integer for the chapter number argument.

User Action: Correct the source file to supply a positive integer.

ARGNOTTYP, at tag or text on line *n* of file
file-spec
Argument *string* to tag <*tagname*> is not valid for doctype *string*

Warning: The indicated argument is not valid for this tag in the indicated doctype.

User Action: Verify that you specified the correct doctype keyword on the DOCUMENT command line.

ARGOUTRAN, at tag or text on line *n* of file
file-spec
Argument *number* is not in the range 0 to 9

Warning: The indicated argument cannot be less than 0 or greater than 9.

User Action: Correct the argument so that the number is in range.

ARGOVRFLW, at tag or text on line *n* of file
file-spec
More than *number* characters in an argument

Fatal: The argument is too large. This often indicates that an argument list was not terminated correctly. The tag translator may be including large portions of the input text as part of the argument.

User Action: Check the tag indicated by the location information and supply the correct termination to the argument list.

AUXOVRFLW, at tag or text on line *n* of file
file-spec
More than *number* auxiliary files open

Fatal: The number of auxiliary files that can be open simultaneously is limited. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Be sure to close an auxiliary file when you have finished reading or writing it.

BADLSTARG, at tag or text on line *n* of file
file-spec
Argument *string* is not valid for <LIST> type *string*

Warning: Verify the argument you specified to the <LIST> tag. Each type of list accepts different keyword arguments, for example, UPPERCASE is a valid keyword for alphabetic lists, but not for numbered lists.

User Action: Verify that you have specified the correct type of list. If you specify a keyword argument, be sure that it is spelled correctly.

BNOTESOUT, at tag or text on line *n* of file
file-spec
Accumulated *string* not referenced.
Notes will not be output

Warning: A document contained <BACK_NOTE> or <BIB_NOTE> tags, but no corresponding <BACK_NOTES> or <BIB_NOTES> was specified to output the accumulated notes.

User Action: Place the <BACK_NOTES> or <BIB_NOTES> tag in the source file at the position at which you want them output.

BOXTOOBIG, at tag or text on line *n* of file
file-spec
Length *number* of boxed item exceeds *number* characters.

Warn: An argument to the <BOX> tag exceeds the maximum number of characters allowed. The text will be output as null.

User Action: Correct the <BOX> tag to specify an argument with fewer characters.

CALL_NEST, at tag or text on line *n* of file
file-spec
Callouts inaccurately nested in a monospaced example.

Error: A monospaced example using callouts incorrectly nests <EXAMPLE> ... <ENDEXAMPLE> tags (or tags that produce monospaced output, such as <CODE_EXAMPLE>) with respect to the <CALLOUTS> ... <ENDCALLOUTS> tags. For example, the following is incorrect:

```
<CALLOUTS>(\PREFIX)
<CODE_EXAMPLE>
  ...
<ENDCALLOUTS>
<ENDCODE_EXAMPLE>
```

User Action: Correct the SDML source file so that the begin and end tags for the example and the callouts are correctly nested, for example:

```
<CALLOUTS>(\PREFIX)
<CODE_EXAMPLE>
  ...
<ENDCODE_EXAMPLE>
<ENDCALLOUTS>
```

CANTOPINT, Cannot open intermediate input file:
file_spec

Fatal: The post translator cannot open the .int_tex file.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to open the file.

CANTOPOUT, Cannot create output file:
file_spec

Fatal: The post translator cannot create the .tex file for some reason.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to create the file.

CASOVRFLW, <CASE> tags have been nested beyond the limit of *number*

Fatal: The <CASE> built-in tag has been nested beyond the allowed limit. Submit a Software Performance Report.

User Action: Verify that the tag definition that uses the <CASE> or <CASE_NUMERIC> tag has been correctly coded.

COMNOTEND, at tag or text on line *n* of file
file-spec
A <COMMENT> in a tag definition has not been terminated

Error: A tag's definition contains a comment that is not properly terminated.

User Action: Examine the tag definition to be certain that the comment text is terminated. If the comment text contains characters such as ampersand, backslash, vertical bar, or parentheses, use the <COMMENT> ... <ENDCOMMENT> format.

CPU_USAGE, Pass 1: *number* Pass 2: *number* Total: *number* seconds

Informational: The tag translator reports its CPU usage during pass 1 and pass 2 and the total for both passes.

User Action: None.

CPYNAMHID, at tag or text on line *n* of file
file-spec
<COPY_TAG> is referencing a tag (<*tagname*>) that is hidden

Warning: The name referenced by the <COPY_TAG> is hidden, and therefore cannot be copied as a new tag definition. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Be sure that the name is spelled correctly and that the sequence of tag executions allows the copy to take place when the tag is not hidden.

CPYNAMUND, at tag or text on line *n* of file
file-spec
<COPY_TAG> is referencing a name (*string*) that is undefined

Warning: The name referenced by the <COPY_TAG> is not defined, and therefore cannot be copied as a new tag definition. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Be sure that the name is spelled correctly and that the tag is already defined.

DEFSLOADD, End of Loading of Tag Definitions

Informational: This message is issued after the tag definitions have been loaded and before the reading of any input files. If any error messages appear before this message, the errors were detected during the loading of the tag definitions.

User Action: None.

DIVBYZERO, at tag or text on line *n* of file
file-spec
Argument 2 to tag <COUNTER> is attempting to divide by zero

Warning: Division by zero is undefined. (Processing continues. The result is as if a divisor of 1 had been used.) If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the correct divisor is specified. You may want to use the <COMPARE_NUMERIC> built_in tag to test the divisor before using it.

DUPHIDNAM, at tag or text on line *n* of file
file-spec
A <HIDE_TAGS> tag is reusing the hide-name, *string*

Warning: The hide-name is already in use. Once tags have been hidden under a specific hide-name that hide-name cannot again be used until the tags have been revealed with the <REVEAL_TAGS> tag. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine the sequence in which <HIDE_TAGS> and <REVEAL_TAGS> tags are invoked using the indicated hide-name. Correct the sequence so that the hide and reveal actions alternate in sequence.

DUPSYMBOL, at tag or text on line *n* of file
file-spec
The symbol *string* is being defined twice.
The earlier definition is replaced by the new definition

Warning: The same symbol is being defined for two different purposes. The tag translator requires that each symbol be unique. If your symbols exceed 31 characters in length, the automatic truncation to 31 characters may be causing two different symbols to look alike.

User Action: Verify that both symbols are defined uniquely within the first 31 characters.

ENDNOTBEG, at tag or text on line *n* of file
file-spec
<*tagname*> specified without corresponding <*tagname*> .

Warn: A terminating tag was specified without the tag that it terminates. The tag is ignored.

User Action: Verify that you correctly entered the beginning tag. Or, remove the extraneous ending tag from your source file.

ENDPASS_1, End of first pass over the input

Informational: The tag translator reports the end of the first pass over all input files. Error messages that appear ahead of this message were issued during the first pass. Error messages that follow this message are issued during the second pass over the input files.

User Action: None.

EOFARGLST, End of file encountered while searching for closing parenthesis. See argument list of tag <*tagname*> on line *number* of file
filename

Fatal: An argument list is not terminated before the end of the current input file. Either a right parenthesis is missing, or an <INCLUDE> tag occurred in a tag's argument list.

User Action: Verify that the indicated tag's argument list is terminated, and that no tag's argument list is unterminated at the end of an included file.

EOFCOLECT, End of file encountered while searching for tag *<tagname>* . See <COLLECT> tag on line *number* of file *filename*

Fatal: A <COLLECT> built-in tag has encountered an end of file before encountering the indicated stop-tag. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify the spelling of the stop-tag argument, and the presence of the stop-tag in the input file.

EOFENDCAS, End of file encountered while searching for tag <ENDCASE> . See <CASE> or <CASE_NUMERIC> tag on line *number* of file *filename*

Fatal: A <CASE> (or <CASE_NUMERIC>) built-in tag is not terminated before the end of file. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that every <CASE> and <CASE_NUMERIC> built-in tag is terminated by an <ENDCASE> tag, even if the <CASE> built-in tags are nested.

EOFENDCOM, End of file encountered while searching for tag <ENDCOMMENT> . See <COMMENT> tag on line *number* of file *filename*

Fatal: Comment text, introduced by the <COMMENT> built-in tag, was not enclosed as an argument, and therefore requires an <ENDCOMMENT> tag as a terminator.

User Action: Verify that the comment text is either enclosed as an argument to the <COMMENT> built-in tag, or that it is terminated with an <ENDCOMMENT> tag.

EOFENDLIT, End of file encountered while searching for <ENDLITERAL> or <ENDDELAYED> . See <LITERAL> or <DELAYED> tag on line *number* of file *filename*

Fatal: Literal text, introduced by the <LITERAL> or <DELAYED> tag, was not enclosed as an argument, and therefore requires an <ENDLITERAL> or <ENDDELAYED> tag as a terminator.

User Action: Verify that the literal text is either enclosed as an argument to the <LITERAL> or <DELAYED> tag, or that it is terminated with an <ENDLITERAL> or <ENDDELAYED> tag.

EOFIGNORE, at tag or text on line *n* of file *file-spec* End of file encountered while searching for tag or label

Fatal: An <IGNORE> built-in tag has encountered the end of the input file without finding one of the tags or labels supplied in the argument list. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify the spelling of the tags and labels in the argument list. You may want to add the label __EOF to the argument list.

ERRCLSINP, at tag or text on line *n* of file
file-spec
Input files not closed at end of pass.

Fatal: Files that were included as part of the input have not been properly closed.

User Action: Submit a Software Performance Report.

ERRDEFLNM, at tag or text on line *n* of file
file-spec
Attempt to define logical name was unsuccessful
Logical: *string*
Equivalence: *string*

Warning: An <INCLUDES_FILE> tag in the <PROFILE> template has failed in its attempt to define the logical name with the given equivalence string.

User Action: Examine the arguments to the <INCLUDES_FILE> tag to ensure that they are a correctly formed name and a file specification. Correct the <INCLUDES_FILE> arguments.

ERRDURGET, at tag or text on line *n* of file
file-spec
Error reading line *number* of file:
filename.
Perhaps the line is too long

Warning: The indicated line probably exceeds the tag translator's buffer size. Processing continues with a truncated line (which may introduce other errors if the information that was lost includes part of an argument list to a tag).

User Action: Edit the input file and shorten the line, or break it into two lines if possible.

ERRDURPUT, at tag or text on line *n* of file
file-spec
Error detected writing output file:
filename

Fatal: An error was detected while attempting to write to the indicated output file.

User Action: Check that your process has sufficient privilege and access rights and that the output device has sufficient free space to hold the file.

ERROPEAUX, at tag or text on line *n* of file
file-spec
Cannot open auxiliary file:
filename

Warning: An attempt to open an auxiliary file failed. The next Informational message may supply the reason for the failure. Processing continues, but attempts to read or write the auxiliary file will fail and will generate error messages. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: If the file is being opened for input, verify that the correct file is being accessed and can be read. If the file is being opened for output, verify

that the file specification is correct, and that you have sufficient resources and access rights to create the file.

ERROPEDMP, at tag or text on line *n* of file
file-spec
Cannot open dump file:
filename

Warning: The <RETRIEVE> built-in tag has failed to open a file for output. The next Informational message may indicate the reason for the failure. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to create the file.

ERROPEINC, at tag or text on line *n* of file
file-spec
Cannot open included file:
filename

Warning: The indicated file cannot be opened for input. The next Informational message may indicate the reason for the failure. Processing continues without the included file.

User Action: Verify that the correct file is being accessed and can be read. If you are using a logical name to access the file, verify that the logical name is defined. If you are relying on an <INCLUDES_FILE> tag to define the logical name, verify that the spelling of the logical name is the same in the <INCLUDES_FILE> tag, and that the <INCLUDES_FILE> tag has been specified for the book element that contains the <INCLUDE> tag.

ERROPEINP, at tag or text on line *n* of file
file-spec
Cannot open input file:
filename

Fatal: The input file specified on the command line cannot be opened. The next Informational message may indicate the reason for the failure.

User Action: Verify that the correct file is being accessed and can be read.

ERROPELOG, at tag or text on line *n* of file
file-spec
Cannot open log file:
filename

Warning: The <SET> built-in tag has failed to open a file for trace output. The next Informational message may indicate the reason for the failure. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to create the file.

ERROPEOUT, at tag or text on line *n* of file
file-spec
Cannot open output file:
filename

Fatal: The tag translator cannot open the output file. The next Informational message may indicate the reason for the failure.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to create the file.

ERROPESEC, at tag or text on line *n* of file
file-spec
Cannot open secondary output file:
filename

Warning: An attempt to open a secondary output file failed. Processing continues, but output will go to the primary output file. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the file specification is correct, and that you have sufficient resources and access rights to create the file.

ERROPESIN, at tag or text on line *n* of file
file-spec
Cannot open cross reference input file:
filename

Fatal: The tag translator failed to open the file that contains the symbols used for cross-references. The next Informational message indicates the reason for the failure.

User Action: The file that contains the symbols used in cross-referencing can be used by only one user at a time. The tag translator locks the file to ensure that users have only serial access to it. If the reason for the failure to open the file was due to the fact that another user was currently using the file, you must wait and reissue the command when the file is no longer in use.

ERROPESOT, at tag or text on line *n* of file
file-spec
Cannot open cross reference output file:
filename

Fatal: The tag translator failed to open a file for writing the symbols used for cross-references. The next Informational message may indicate the reason for the failure.

User Action: Verify that you have sufficient resources and access rights to create the file.

ERRPRSFSP, at tag or text on line *n* of file
file-spec
Error parsing file specification:
filename

Warning: The file specification contains an error.

User Action: Correct the file specification or logical name definition.

ERRVMZONE, at tag or text on line *n* of file *file-spec*
Internal error. Failure to create VM zone.

Fatal: The tag translator failed to create space in virtual memory for storage of data.

User Action: Submit a Software Performance Report.

EXAMINLIN, Examine line *number* of file *filename*

Informational: This message follows other messages of greater severity, and directs you to a possible error in the input.

User Action: Use the information to examine and correct your input.

EXFOOTNOT, at tag or text on line *n* of file *file-spec*
Footnotes in a monospaced example exceed maximum of 4.

Warning: You have specified more than four <FOOTNOTE> tags in a <CODE_EXAMPLE> or other monospaced example. The maximum allowed is four. The footnote will not be processed.

User Action: Correct the source file by removing the extraneous footnotes.

EXPAPPLET, Explicit appendix letter *string* set on line *number* of file *filename*

Informational: This message warns that the letter assigned to the next <APPENDIX> tag has been supplied explicitly by a <SET_APPENDIX_LETTER> tag. The appendixes may be lettered out of sequence.

User Action: No action is required.

EXPAPPNUM, Explicit appendix number *number* set on line *number* of file *filename*

Informational: This message warns that the number assigned to the next <APPENDIX> tag has been supplied explicitly by a <SET_APPENDIX_NUMBER> tag. The appendices may be numbered out of sequence.

User Action: No action is required.

EXPCHAPNO, Explicit chapter number *number* set on line *number* of file *filename*

Informational: This message warns that the number assigned to the next <CHAPTER> tag has been supplied explicitly by a <SET_CHAPTER_NUMBER> tag. The chapters may be numbered out of sequence.

User Action: No action is required.

FILEWRTNG, File *filename* written (0 length)

Informational: An Error or Fatal message has been issued during the tag translation of a book element. The output file for that book element is reduced to 0 length.

User Action: Correct the problems indicated in the Error and Fatal messages and reprocess.

FILEWRTOK, File *filename* written

Informational: An element of a book has been processed through the tag translation phase.

User Action: None.

FCMDPARMS, at tag or text on line *n* of file
file-spec
<*tagname*> specified without <*tagname*> in Format.
Using <*tagname*> alone

Warning: The tags <FCMD> and <FPARMS> must be used together to produce predictable results.

User Action: Modify your source file so that it contains both the <FCMD> and <FPARMS> tags. Arguments to either of these tags may be null, that is, it is valid to specify <FCMD> .

FIG_DEPTH, at tag or text on line *n* of file
file-spec
Tag <*tagname*> value *number* exceeds the maximum *NUMBER* or is 0.

Warning: The indicated tag specifies a depth for a figure or space to be left for a figure that will overrun the page boundaries. Or, the argument was specified as 0. In either case, the document's default full page depth will be used.

User Action: Correct the tag so that you specify no more than the maximum allowed.

FIG_WIDTH, at tag or text on line *n* of file
file-spec
Tag <*tagname*> value *number* exceeds the maximum *NUMBER* or is 0.

Warning: The indicated tag specifies a width for a figure or space to be left for a figure that will overrun the page boundaries. Or, the argument was specified as 0. In either case, the document's default full page width will be used.

User Action: Correct the tag so that you specify no more than the maximum allowed.

FIGINDENT, at tag or text on line *n* of file
file-spec
The value *number* specified for a block indent exceeds the current allowed maximum of *number*.

Warning: A <FIGURE_FILE> tag specified the INDENT argument and a value that exceeds the current maximum allowed. The maximum value will be used.

User Action: Correct the numeric argument specified with INDENT so that it does not exceed the indicated maximum.

FIGLINMAX, at tag or text on line *n* of file
file-spec
Monospaced example lines in <*tagname*> exceed maximum *number*.

Error: A monospaced example specified in the context of a <FIGURE> or <EXAMPLE> tag is too long to fit on the current page and either no <VALID_BREAK> tags were specified to provide valid break points, or there are too many lines between <VALID_BREAK> tags.

User Action: Put <VALID_BREAK> tags at suitable places in the monospaced example to allow the pages to break automatically.

FILENUMNG, at tag or text on line *n* of file
file-spec
File number is out of range

Warning: A number supplied to the <FILE_NAME> tag is not within the range of 1 to the number of files that have been opened. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Do not attempt to pass a number to the <FILE_NAME> tag unless that number was previously returned by an invocation of the <FILE_NUMBER> tag. This guarantees that only legal file numbers will be used as arguments to the <FILE_NAME> tag.

FILISOPEN, at tag or text on line *n* of file
file-spec
Auxiliary file:
filename
is already open

Warning: An <OPEN> built-in tag is specifying a file that is already open. The request is ignored, and processing continues. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Check the file specification to be sure that the correct file is being specified. Remove the second request to open the same file.

FILNOTOPN, at tag or text on line *n* of file
file-spec
Auxiliary file:
filename
is not open

Warning: A <READ> or <WRITE> built-in tag is addressing a file that has not been opened. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Verify that the file is being correctly referenced and that the file has been opened with the <OPEN> built-in tag.

FMTDEVICE, Tag *<tagname>* produces device-specific output.

Informational: A tag that produces specific formatting controls has been processed. The output may not be suitable if the file is processed for another document type, destination, or both. For example, <FINAL_CLEANUP> (LINE_BREAK) tells the text processor to create a new line of output, but the line break may not be suitable on all output devices.

User Action: Use explicit formatting commands sparingly.

GTMAXARGS, at tag or text on line *n* of file
file-spec
More than *number* arguments supplied to tag *<tagname>*

Error: The indicated tag does not expect more than the indicated number of arguments. Perhaps you included a backslash character in one of the arguments, which the tag translator interpreted as an argument separator.

User Action: Verify that the argument list is correctly coded. If necessary, use the <BACKSLASH> tag to code a backslash character that is actually part of an argument.

HIDNOTHID, at tag or text on line *n* of file
file-spec
Internal error. A hidden tag was invoked,
but was not found in the data structure of hidden tag names

Fatal: The algorithm for hiding and revealing tags has failed. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Submit a Software Performance Report.

ICON_TEXT, at tag or text on line *n* of file
file-spec
No text supplied for <ICON> .

Warning: An <ICON> tag was specified, but no <ICON_TEXT> tag specified text to accompany the graphics. No text will be output.

User Action: Verify that you specified <ICON_TEXT> correctly.

IGNAMEILL, at tag or text on line *n* of file
file-spec
Argument *string* to <IGNORE> is illegal

Warning: The argument is not a legal name or label. The argument is dropped from the argument list, and processing continues. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine the argument list to the <IGNORE> tag.

IGNAMEMIS, at tag or text on line *n* of file
file-spec
An argument to <IGNORE> tag is a null string

Warning: One of the arguments to an <IGNORE> tag is empty. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine the argument list to the <IGNORE> tag. The tag names should not have brackets around them.

IGNORESET, Ignoring <SET_CHAPTER_NUMBER> or <SET_APPENDIX_LETTER> on line *number* of file
filename

Informational: The <SET_CHAPTER_NUMBER> and <SET_APPENDIX_LETTER> tags are ignored when doing a book build or element build, because the numbering of book elements is done automatically.

User Action: Remove the tag.

IGNOTDONE, at tag or text on line *n* of file
file-spec
The <IGNORE> tag has no arguments, so it is ignored

Warning: Missing or illegal arguments have resulted in an <IGNORE> tag that has no legal arguments. The <IGNORE> tag is not executed. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine the argument list to the <IGNORE> tag.

INCNOTARG, at tag or text on line *n* of file
file-spec
An <INCLUDE> tag cannot be invoked in an argument

Error: An <INCLUDE> tag can be invoked only in text.

User Action: Move the <INCLUDE> tag outside the argument list.

INCOVRFLW, at tag or text on line *n* of file
file-spec
More than *number* nested levels of included files

Fatal: The number of included files that can be open at the same time is limited. The limit has been exceeded.

User Action: Consider whether the included files can be included sequentially rather than simultaneously. If file A includes file B, and file B includes file C, consider whether file A can include both B and C, one after the other.

INDENTVAL, at tag or text on line *n* of file
file-spec
The *<tagname>* specifies an indent value of *number*.
This exceeds the maximum *number* allowed for this tag.

Warning: The INDENT argument to the specified tag specified an indent value that exceeds the maximum allowed. The maximum value will be used.

User Action: Correct the tag to specify an indent value less than or equal to the maximum allowed.

INT_LOGIC, at tag or text on line *n* of file
file-spec
Internal error processing tag <NAME>

Error: An SDML tag definition has an error in it.

User Action: Submit a Software Performance Report.

ISTHISTAG, at tag or text on line *n* of file
file-spec
Ignoring <*string*. Is this a tag without a closing angle bracket?

Warning: An apparent tagname has been found, but without the closing angle bracket that is needed to make it a complete tag invocation. This sometimes occurs when an underscore is typed as a hyphen. For example, <TABLE_SETUP> is typed as <TABLE-SETUP>. The tag translator stops on the hyphen because that is an illegal character in a tagname, and it assumes that you may have meant to type <TABLE> -.

User Action: If this is a typo, simply add the closing angle bracket or change the hyphen to an underscore. If this is not a typo (you really want the left angle bracket and the tagname to appear in the output), use the <LITERAL> tag to enclose the left angle bracket. This action will suppress the warning message.

KEYPADROW, at tag or text on line *n* of file
file-spec
Too many keypad rows. Extra rows ignored

Warning: A set of tags within a <KEYPAD_SECTION> specifies more than four <KEYPAD_ROW> tags. The keyboard keypad has only four rows and one end row.

User Action: Determine which <KEYPAD_ROW> tag is extra and remove it from your source file.

LASTAGWAS, Last occurrence of <*tagname*> was on line *number*

Informational: This message is issued when a tag is not ended before a particular context or the end-of-file. It tells you the line number of the last occurrence of the tag.

User Action: Use the indicated line number to determine where to correct your source file.

LISTYPWAS, List type was *string*

Informational: This message accompanies an error that indicates that a <LIST> tag was not terminated. It indicates the type of list, for example, NUMBERED, UNNUMBERED, and so on.

User Action: Use this information to correct your SDML source file.

LITOVRFLW, More than *number* characters in literal. See line *number* of file *filespec*

Fatal: The size of the text encompassed by a <LITERAL> or <DELAYED> tag is limited. This error may indicate that a <LITERAL> or <DELAYED> tag was incorrectly terminated.

User Action: Verify that the <LITERAL> or <DELAYED> tag has been correctly coded. If the text to be encompassed by the tag exceeds the limit, you must break it up and use more than one <LITERAL> or <DELAYED> tag.

LTMINARGS, at tag or text on line *n* of file
file-spec
Fewer than *number* arguments supplied to tag *<tagname>*

Error: The indicated tag requires a minimum number of arguments. Perhaps one of the arguments has an unmatched right parenthesis in it, and the tag translator has treated that character as the terminator of the argument list.

User Action: Review and correct the argument list. Be sure that the open parenthesis for the argument list is not preceded with a space, for example <TAG> (ARG) should be <TAG>(ARG).

MARKUNBAL, at tag or text on line *n* of file
file-spec
<MARK> tags are unbalanced.

Error: A <MARK> tag and its corresponding <ENDMARK> tag cross a tag's boundary or are incorrectly nested with respect to another pair of tags. For example, if a <MARK> tag is specified before <FORMAT> and <ENDMARK> is specified before <ENDFORMAT>, the tags are considered unbalanced.

User Action: Look at the source file location of the indicated tag, and move it. In particular look for instances where a <MARK> tag is specified in a tag argument, but the <ENDMARK> tag is specified outside that tag's argument or in an argument to another tag.

MINEXCMAX, at tag or text on line *n* of file
file-spec
Minimum argument count (*number*) exceeds maximum argument count (*number*).
(Minimum argument count is being reset to zero)

Warning: A <DEFINE> or <REDEFINE> tag is specifying a minimum and maximum number of arguments and the minimum exceeds the maximum. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Correct the minimum and maximum argument values.

MSGIDISNG, at tag or text on line *n* of file
file-spec
The message identification *string* is invalid.
The <HIDE_TAGS> tag is ignored

Warning: The second argument to a <HIDE_TAGS> tag is supplying an unknown message identification. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Check the spelling of the message identification argument to be sure that it agrees with one of the error messages in the message file.

NAMETRUNC, at tag or text on line *n* of file
file-spec
Name *string* exceeds 31 characters.
Name *string* is truncated version

Warning: Names may not exceed 31 characters in length. The name that was supplied is a legal name, but it is too long. It is truncated to 31 characters.

If more than one name is truncated, the shortened names may not be unique and other errors may occur depending on the type of name.

User Action: Shorten this name (and any other names that are too long) to 31 characters or less, so that there is no potential for other errors.

NAME2LONG, at tag or text on line *n* of file
file-spec
Condition name *string* exceeds 28 characters

Error: Condition names may not exceed 28 characters in length.

User Action: Shorten this name (and any other condition names that are too long) to 28 characters or less.

NESTEDCOM, A <COMMENT> tag on line *number* of file
filename
is nested within <COMMENT> ... <ENDCOMMENT> that starts on line *number* of file
filename

Warning: <COMMENT> tags cannot be nested.

Sometimes an error is made in typing a <COMMENT> tag in the format <COMMENT> (text). The typo makes it appear that the tag does not have an argument list, and it is assumed that an <ENDCOMMENT> tag is present to end the comment text. During the search for the <ENDCOMMENT> tag, if another <COMMENT> tag is encountered it will appear to be nested within the first <COMMENT> tag's text.

User Action: Correct any typographical errors in the initial <COMMENT> tag or remove the nested <COMMENT> tag.

NODEFITEM, at tag or text on line *n* of file
file-spec
Definition list specified no items before <*tagname*> .

Warning: A <DEFINITION_LIST> tag, or a corresponding tag in the SOFTWARE doctype, was specified but no <DEFLIST_ITEM> / <DEFLIST_DEF> tags were specified for the list. No output will be produced.

User Action: Remove the definition list start/end tags; or provide item text for the definition list.

NOENDPROF, at tag or text on line *n* of file
file-spec
The <PROFILE> tag was not terminated by an <ENDPROFILE> tag

Warning: In order for the bookbuild process to be completed, the <PROFILE> tag must be terminated by an <ENDPROFILE> tag. The <ENDPROFILE> tag is supplied by default in this case.

User Action: If the <PROFILE> tag is not terminated by an <ENDPROFILE> tag, add one after the final <ELEMENT> or <INCLUDES_FILE> tag.

NOFIGELMS, at tag or text on line *n* of file
file-spec
<*tagname*> has no elements.

Warning: A <FIGURE> or <EXAMPLE> tag was specified and terminated but there were no elements (for example, <FIGURE_SPACE>), to place in the figure.

User Action: Verify that you correctly spelled and entered tags for the figure or example.

NOHIDERVL, at tag or text on line *n* of file
file-spec
Ignoring request to hide the <REVEAL_TAGS> tag

Warning: The <REVEAL_TAGS> built-in tag cannot be hidden by the action of a <HIDE_TAGS> built-in tag. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Remove the word REVEAL_TAGS from the argument list of the <HIDE_TAGS> tag.

NONESTING, at tag or text on line *n* of file
file-spec
Invalid attempt to nest tag <*tagname*> . Ignored

Warning: A tag is referenced when its context is already active. For example, this message is issued when <CODE_EXAMPLE> is specified when a previous <CODE_EXAMPLE> is not terminated.

User Action: The indicated tag is ignored; however, you should verify your source file to determine whether the tag is a duplicate or whether, in fact, you did not terminate an earlier occurrence of the same tag.

NONEWFILE, at tag or text on line *n* of file
file-spec
<*tagname*> specified without *string*. No new file will be generated.

Warning: A <CONTENTS_FILE> or <INDEX_FILE> was encountered in the input file, but the corresponding /CONTENTS or /INDEX qualifier was not present on the command line. No new contents or index will be produced. An earlier version of a contents or index file, if it exists, may be included in the output.

User Action: If you want to include the current table of contents or index file, you should reissue the DOCUMENT command using the appropriate qualifier.

NOPROFILE, at tag or text on line *n* of file
file-spec
Tag can appear only following a <PROFILE> tag

Fatal: The tag is part of a profile and can be used only in that context and only within the same file as the <PROFILE> tag.

User Action: Check that the tag appears after a <PROFILE> tag, and in the same file as the <PROFILE> tag.

NOREADOUT, at tag or text on line *n* of file
file-spec
Cannot read from file:
filename
The file is opened as output

Warning: A <READ> built-in tag cannot be addressed to an auxiliary file that is opened as output. (The <OPEN> tag defaults to output mode unless INPUT is specified.) If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Correct the <OPEN> built-in tag to open the file as input.

[redacted]
Tag <*tagname*> is not allowed in the *string* doctype

Error: The indicated tag is not meaningful in the context of the doctype specified on the command line and cannot be processed. For example, the LETTER doctype does not allow <CHAPTER> or heading-level tags.

User Action: Verify that you specify the correct doctype keyword on the DOCUMENT command line. If the doctype is correct, remove the tags from the SDML source file.

NOTHIDNAM, at tag or text on line *n* of file
file-spec
A <REVEAL_TAGS> tag is referencing an unknown hide-name,
string

Warning: The hide-name specified by a <REVEAL_TAGS> tag is not in use. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine the sequence in which <HIDE_TAGS> and <REVEAL_TAGS> tags are invoked using the indicated hide-name. Correct the sequence so that the hide and reveal actions alternate in sequence.

NOTINELEM, at tag or text on line *n* of file
file-spec
Tag can appear only within the context established by
an element heading tag such as <CHAPTER> , <PART> , etc

Error: When a bookbuild is done, all text and the tags that accompany the text must appear within some element of the book. The text and tags must follow the element heading tag in order to be considered to be in the context of the element.

User Action: Correct the source text so that the tag that is in error follows a tag that heads an element of the book.

NOTINPROF, at tag or text on line *n* of file
file-spec
The *<tagname>* tag must appear in the same file as the
<PROFILE> tag

Error: All the tags that make up a profile must appear in the same file.

User Action: Move the tag into the same file as the <PROFILE> tag.

NOZONEYET, at tag or text on line *n* of file
file-spec
Tag *<tagname>* specified outside its valid zone or context.
It must be preceded by tag *<tagname>* .

Error: A tag was specified outside the context that enables it, for example, a <COPYRIGHT_PAGE> tag was specified when the tag <FRONT_MATTER> has not been specified to enable the front matter zone.

User Action: Be sure that you have specified the correct tag. Add the appropriate zone-enabling tag.

NOWRITEIN, at tag or text on line *n* of file
file-spec
Cannot write to file:
filename
The file is opened as input

Warning: The <WRITE> built-in tag cannot write to an auxiliary file that has been explicitly opened as input. If you receive this message and are under a

service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Correct the <OPEN> built-in tag or use the proper tag (<READ> or <WRITE>) to agree with the mode in which the file has been opened.

NOUNDEFBI, at tag or text on line *n* of file
file-spec
Ignoring <UNDEFINE> or <REDEFINE> of the built-in definition of *<tagname>*

Warning: A tag translator built-in tag cannot be deleted by the action of a <REDEFINE> or <UNDEFINE> . The built-in tag can be "stacked" by issuing a <DEFINE> tag for it, however. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Change a <REDEFINE> to a <DEFINE> . Change an <UNDEFINE> to a <DEFINE> that has an empty definition.

NUMBERARG, at tag or text on line *n* of file
file-spec
Argument *number* is not valid for *<tagname>* .

Warning: The numeric argument is not valid for the tag.

User Action: Consult the tag's documentation to determine the valid values.

NUMLTZERO, The argument is less than 0.

Informational: This message accompanies a message about an invalid numeric argument; it indicates that a negative value was specified for a tag argument which must be a positive number.

User Action: Consult the tag's documentation to determine the valid values.

OTL_TITLE, at tag or text on line *n* of file
file-spec
<tagname> not valid within outlines.
Enter the title as an argument to the <OUTLINE> tag

Warning: A <TITLE> tag was specified within the context of the <OUTLINE> tags.

User Action: Correct your source file so that the title of the outline is supplied in arguments to the <OUTLINE> tag.

OTL_LEVEL, at tag or text on line *n* of file
file-spec
Outline level specified, *number,* is not in range
The argument must specify 1 through 6

Warning: An invalid number was specified in a <LEVEL> tag in an outline. Only levels of 1 through 6 are valid.

User Action: Correct your source file so that no outline level has a number greater than 6.

OUTLIN2BG, at tag or text on line *n* of file
file-spec
Output line exceeded *number* bytes, and had to be broken

Warning: Text in an output line exceeded the tag translator's buffer. No text was lost because the line was written as two lines, but the division into two lines may have broken a word into two words, or interfered with the correct formatting of the text. This usually happens when the tag translator inserts a great many formatting commands into an output line. It also happens when a tag reference on a long input line is replaced by a very long string of text.

User Action: Check the formatted output that corresponds to the input line specified in the location information to see if breaking the line affected the formatting.

POSTERROR, at tag or text on line *n* of file
file-spec
Error encountered during post-translation

Fatal: An error has been detected during the final step in tag translation.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

PQLISNONE, at tag or text on line *n* of file
file-spec
<*tagname*> specifies 'NONE' when no heading is specified

Warning: The keyword argument "NONE" was specified to a <PARAMDEFLIST> or <QUALDEFLIST> tag in a context in which no default heading is in effect. The use of NONE in this context is meaningless.

User Action: Determine if you intended to be using default headings for the <PARAMDEFLIST> or <QUALDEFLIST> tag. If not, remove the tag from your source file. Otherwise, verify whether you should have invoked one of the reference templates to set an environment for parameter/qualifier list descriptions.

PROFORDER, at tag or text on line *n* of file
file-spec
<PROFILE> tag is out of order.

Fatal: A <PROFILE> tag has been detected after an element heading tag, such as <CHAPTER>.

User Action: Place a single <PROFILE> ... <ENDPROFILE> sequence in a file by itself in order to do a bookbuild.

PROFQ_ILL, at tag or text on line *n* of file
file-spec
/PROFILE qualifier illegal during a bookbuild

Fatal: The /PROFILE qualifier was supplied on the command line indicating that an element build is desired, but the SDML file contains a <PROFILE> tag, indicating that a bookbuild is desired. The /PROFILE qualifier (which indicates an element build) should not be supplied when a bookbuild is desired.

User Action: Invoke DOCUMENT without the /PROFILE qualifier.

QUALVALUE, at tag or text on line *n* of file
file-spec
Value *string* to /CONDITION qualifier is not a valid name

Fatal: The condition name supplied with the /CONDITION qualifier must contain only alphabetic and numeric characters and the underscore character.

User Action: Correct the spelling of the condition name.

REV_PAGES, Update pages *number* through *number* will be output.

Informational: This message indicates the pages that will be output for an update.

User Action: None.

REVISINFO, Document is using revision indicators

Informational: This message indicates that the <REVISION> tag is being processed. Text that is delimited by <MARK> and <ENDMARK> tags will be accompanied by revision indicators on output.

User Action: If you intend for revision bars to print in your text, you do not need to take any action. However, if you intended for text indicated by <MARK> tags to not appear in the output, you should remove the <REVISION> tag from the file.

REVISTEXT, Revised pages will contain the text '*string*'.

Informational: This message verifies the text specified in the second argument to the revision tag and reminds you that only pages specified in an update range will be processed and printed.

User Action: None.

REVIS_UPD, Revision will contain update pages only.

Informational: This message reminds you that you specified <REVISION> (UPDATE) and that your output file will only contain the text for pages specified between <UPDATE_RANGE> and <ENDUPDATE_RANGE> tags.

User Action: None.

RMVCONDTN, Removing condition *condition-name*
on line *number* of file *filespec*

Informational: A <SET_CONDITION> (REMOVE) tag is being executed.

User Action: None.

SAVTAGACT, at tag or text on line *n* of file *file-spec*
Tag <*tagname*> has a before or after action that is ignored when saving a tag table

Warning: You are attempting to invoke a tag that has a before action or an after action. The before action or after action cannot be executed while you are saving the tag in a tag table. The before action or after action will be executed when the saved tag table is loaded. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: No action. However, you may want to verify that this tag should be invoked when the saved tag table is actually loaded.

SAVTAGILL, at tag or text on line *n* of file *file-spec*
This built-in tag cannot be invoked during the saving of a tag table

Error: Only certain built-in tags can be stored in a saved tag table for invocation when the saved tag table is loaded. This tag is not one that can be saved. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Examine your need to invoke this tag when the tag table is loaded. If you still need to invoke the tag, store it as a quoted string in a string variable. Then reference the string variable. Here is an example:

Suppose that you want to invoke the <FILE_NAME> tag during the loading of the saved tag table. You must store the <FILE_NAME> tag in a string and then output the string.

```
<STRING>(my_file_name\|<FILE_NAME>&)
<STRING>(my_file_name)
```

This works because the <STRING> tag is one of the tags that can be saved in the saved tag table.

SAVTAGLEV, at tag or text on line *n* of file *file-spec*
Tag <*tagname*> cannot be invoked within an argument list during saving of a tag table

Error: During the saving of a tag table, the arguments to all tags must be constant or must be made constant by quoting them. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Place a vertical bar before the first character of the argument and an ampersand after the last character of the argument.

SDML_INFO, *message*

Informational: An SDML tag is describing action taken during tag loading.

User Action: None. The message is purely informational.

SECFILOPN, at tag or text on line *n* of file
file-spec
Cannot open a secondary output file named
filename
A secondary output file is already open. It is named
filename

Error: Only one secondary output file can be open at a time. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: Use the <CLOSE_SECONDARY_OUTPUT> tag to close any secondary output file before using the <OPEN_SECONDARY_OUTPUT> tag to open a new file.

SETCONDTN, Setting condition *condition-name*
on line *number* of file *filespec*

Informational: A <SET_CONDITION> tag is being executed or a condition is being set in response to a /CONDITION qualifier on the command line.

User Action: None.

SKIPITEMS, at tag or text on line *n* of file
file-spec
number *<tagname>* outside of *<tagname>* skipped.

Warning: This message is usually preceded by a message indicating that an <LE> or <TABLE_ROW> tag was encountered; it indicates the number of additional tags that were specified. The tags will not be processed, and will be written, unformatted, to the output file.

User Action: Determine if the tags are considered invalid because there is no preceding <TABLE> or <LIST> tag, or if the <TABLE> or <LIST> tag was specified incorrectly. Correct your source file.

STKOVRFLW, at tag or text on line *n* of file
file-spec
More than *number* nested levels of tags. (These are
tags that are invoked inside other tags' argument lists)

Fatal: The tag translator limits the number of tags that can be invoked simultaneously. Tag B is invoked simultaneously with tag A, if it is invoked in the argument list to tag A. Thus, <A> (<B> (<C>)) requires that three tags are being invoked simultaneously. This may occur unexpectedly when the argument list to one of the tags is not properly terminated.

User Action: Verify that the argument list of the tag that is specified in the location information has been properly terminated.

STRFREEBA, at tag or text on line *n* of file
file-spec
Internal error. Attempt to free string with bad address

Fatal: The tag translator's memory allocation algorithm has failed.

User Action: Submit a Software Performance Report.

STRFREE2T, at tag or text on line *n* of file
file-spec
Internal error. Attempt to free the same string twice

Fatal: The tag translator's memory allocation algorithm has failed.

User Action: Submit a Software Performance Report.

STRTOOBIG, at tag or text on line *n* of file
file-spec
Internal error. Attempting to allocate *number*-byte string

Fatal: The tag translator's memory allocation algorithm has failed.

User Action: Submit a Software Performance Report.

SYMFREEBA, at tag or text on line *n* of file
file-spec
Internal error. Attempt to free symbol table entry with bad address

Fatal: The tag translator's memory allocation algorithm has failed.

User Action: Submit a Software Performance Report.

SYMISSING, at tag or text on line *n* of file
file-spec
Missing symbol argument.
Each book element must have a symbol argument

Fatal: Each element of a book must have a unique symbol so that the element can be processed independently. The symbol table cannot be saved from this run.

User Action: Supply a symbol as an argument to the tag.

SYMISUSED, at tag or text on line *n* of file
file-spec
The symbol *string* is already in use as a symbol of type *string*

Error: Each element of a book must have a unique symbol. You have supplied a symbol on this element of the book that conflicts with an earlier use of this symbol. The symbol table cannot be saved from this run.

User Action: Correct the use of symbols on elements of the book so that each element has a unique symbol. If you are doing a bookbuild you must repeat the bookbuild in order to produce a good symbol table. If you are doing an element build, just correct the symbol in this tag so that this element of the book has a unique symbol and then repeat the element build.

SYMNOTDEF, at tag or text on line *n* of file
file-spec
Symbol *string* is undefined

Warning: A symbol has been referenced but is not in the symbol table.

User Action: Verify that the symbol is spelled correctly and that the same symbol has been entered in the symbol table by the invocation of a tag that defines the symbol.

TAG_FAILS, The tag translator has detected a fatal error

Error: The tag translator is terminating execution with failure status. The accompanying error message supplies details of the error.

User Action: See the description of the accompanying error message.

TAGFREEBA, at tag or text on line *n* of file
file-spec
Internal error. Attempt to free tag table entry with bad address

Fatal: The tag translator's memory allocation algorithm has failed.

User Action: Submit a Software Performance Report.

TAG_INARG, at tag or text on line *n* of file
file-spec
<*tagname*> is invalid in an argument to <*tagname*> .

Fatal: The indicated tag was specified in an argument to a tag that does not allow it. For example, a header level tag was specified in an argument to <TABLE_ROW> .

User Action: Verify that you correctly terminated the argument list for the tag in which this tag was specified.

TAGINFILE, The tag was in the file *filename*

Informational: This message may accompany a message indicating that a tag that must be ended was not terminated. This message is output when the starting tag is not in the current input file.

User Action: Use the file-spec given to determine which file contains the tag that was not properly terminated and terminate it.

TAGINVALD, at tag or text on line *n* of file
file-spec
Tag <*tagname*> is invalid in this context.
The context was established by <*tagname*> on line *number* of *filename*

Error: A tag was specified in a context where it is not valid, for example, a header-level tag was specified within the context of a monospaced example. The message tells you what tag established the context.

User Action: Determine the current context at the point at which the message was issued to determine if you have forgotten a required terminator and correct your source file.

TAGINZONE, at tag or text on line *n* of file
file-spec
Tag <*tagname*> is invalid in the bounds of *string*.

Error: A tag was specified in a context where it is not valid, for example, a header-level tag was specified within the context of a monospaced example.

User Action: Determine the current context at the point at which the message was issued to determine if you have forgotten a required terminator and correct your source file.

TAGNOTDEF, at tag or text on line *n* of file
file-spec
Tag <*tagname*> is undefined

Warning: A tag has been invoked, but no definition exists for the tag. This message appears for the first invocation of an undefined tag. Subsequent invocations of the same undefined tag are counted, but the message is not repeated. A count of the total number of all invocations of undefined tags is reported at the end of the tag translator's processing.

User Action: Correct the spelling of the tag, or supply a definition for the tag.

TAGNOTEND, at tag or text on line *n* of file
file-spec
Tag <*tagname*> from line *number* not terminated

Error: A tag that has a required terminator was not terminated.

User Action: Locate the tag that was not terminated and include its terminating tag at the appropriate position in your source file.

TAGNOTHID, at tag or text on line *n* of file
file-spec
Cannot reveal tag <*tagname*> . It is not hidden

Warning: A <REVEAL_TAGS> tag is requesting that a hidden tag be revealed, but the tag is not currently hidden. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

User Action: After a tag is hidden through the action of a <HIDE_TAGS> tag, it can only be revealed through the action of a <REVEAL_TAGS> tag. If the tag is given a new definition (using the <DEFINE> tag) while it is hidden, it will not appear to be hidden when the <REVEAL_TAGS> tag is executed, and this error message will occur. Check the sequence of your use of the <HIDE_TAGS> , <DEFINE> , and <REVEAL_TAGS> tags.

TAGNOTPRE, at tag or text on line *n* of file
file-spec
Tag <*tagname*> invalid unless preceded by *string*

Warn: A tag is being invoked that is invalid because it is being used out of context.

User Action: Check the sequence of tags to ensure that the proper context is established before the tag is used. Correct the SDML source file.

TBLATTRIB, at tag or text on line *n* of file
file-spec
Argument *string* to tag <*tagname*> is not a valid table attribute

Warn: A keyword was specified in a <TABLE_ATTRIBUTES> tag that is not a valid keyword attribute for tables.

User Action: Verify that the specified value is a valid table attribute and check its spelling. Correct the SDML source file.

TBLATTIGN, at tag or text on line *n* of file *file-spec*
<TABLE_ATTRIBUTES> specified following <TABLE_SETUP> ignored.

Warning: The <TABLE_ATTRIBUTES> tag must be specified preceding the <TABLE_SETUP> tag. The specified attributes will be ignored.

User Action: Move the <TABLE_ATTRIBUTES> tag in front of the <TABLE_SETUP> tag.

TBLBRKCOL, at tag or text on line *n* of file *file-spec*
Table cannot be broken in column *number*.

Warning: The <VALID_TABLE_ROW_BREAK> tag is valid only in columns 2 through 4 of a table.

User Action: Remove the <VALID_TABLE_ROW_BREAK> tag. If the text in the column is too long for a page, consider restructuring the table to be less complex.

TBLCOLCNT, at tag or text on line *n* of file *file-spec*
Tag <TABLE_SETUP> argument 1 '*string*' is missing or is not a number in the range 2 through 9

Error: The <TABLE_SETUP> tag must specify a numeric value in the range of 2 through 9 to specify the number of columns in the table. The indicated argument is invalid.

User Action: Correct the <TABLE_SETUP> tag.

TBLCOLNUM, at tag or text on line *n* of file *file-spec*
Table column size argument '*string*' is missing or is not numeric

Error: An argument specified as a width for table rows in setting up the table is not a numeric argument.

User Action: Correct the <TABLE_SETUP> tag that specifies a non-numeric value for a table column width.

TBLCOLWID, at tag or text on line *n* of file *file-spec*
Sum of table column widths *number* is too large or is 0 or less.

Error: The sum of the values specified for the column widths in the <TABLE_SETUP> tag is excessive, or is less than or equal to 0. The message displays the current summed value of all column widths specified.

User Action: Verify that the numbers you specified in <TABLE_SETUP> are accurate. Adjust them to smaller numbers so that the table will fit. If you did not specify <TABLE_ATTRIBUTES> (WIDE) and the document design you are using provides an extra margin for wide tables, specify <TABLE_ATTRIBUTES> (WIDE) in front of the <TABLE_SETUP> tag. To request that the table appear in a smaller typesize, use <TABLE_ATTRIBUTES> (MAXIMUM).

TBLDUPSET, at tag or text on line *n* of file
file-spec
Tag <TABLE_SETUP> specified twice. Duplicate setup ignored.

Warning: A table contains two <TABLE_SETUP> tags. The second tag is being ignored.

User Action: Determine which <TABLE_SETUP> tag is correct and remove the extra one.

TBLFNOTEM, at tag or text on line *n* of file
file-spec
Table footnote number *number* exceeds maximum *number*

Warning: A numeric argument specified for a footnote declaration in a table exceeds the maximum. The maximum number of table footnotes is normally 12; however, if you use any non-numeric keywords in table footnotes, the maximum number allowed is reduced to 6.

User Action: Verify the number of footnotes in the table, and the numeric arguments you use to declare and reference them.

TBLKEYDUP, at tag or text on line *n* of file
file-spec
Table already specifies <TABLE_KEY> . Duplicate ignored.

Warning: Only one key can be specified in a table. The second key specified is being ignored.

User Action: Determine which table key is the one that you want and remove the duplicate one.

TBLKEYNES, at tag or text on line *n* of file
file-spec
<tagname> is not valid in a nested table.
Nested table on line *string* in file *string*.

Warning: A <TABLE_KEY> or <TABLE_KEYREF> tag was specified in a nested table, that is a table that is specified in another table's <TABLE_ROW> . Table keys and table key references are invalid in nested tables.

User Action: Place the table key and/or the reference to it in the outer table.

TBLKEYNUM, at tag or text on line *n* of file
file-spec
The <TABLE_KEY> tag will replace table footnote 12.

Warning: When a key is specified in a table, you cannot specify a table footnote numbered 12. The text of the table key will replace the footnote number 12, and references to footnote 12 will result in the table key being printed.

User Action: Check the arguments you specified to the <FOOTNOTE> tags in the table and do not specify a number greater than 11.

TBLKNOREF, at tag or text on line *n* of file *file-spec*
Tag <TABLE_KEYREF> tag encountered in a table that has no key.

Warning: A <TABLE_KEYREF> tag is not valid because there is no key in the table to reference.

User Action: Verify that the <TABLE_KEY> tag is correctly specified in the main table (it must not be specified in a nested table). If the table has no key, remove the <TABLE_KEY> tag.

TBLNBREAK, at tag or text on line *n* of file *file-spec*
A nested table with more than three columns cannot be broken.

Error: The <NESTED_TABLE_BREAK> was specified in a nested table that has more than three columns.

User Action: Verify that the <TABLE_SETUP> tag for the table you are trying to break was correctly specified. If the nested table has four columns, you may need to restructure your information.

TBLNONEST, at tag or text on line *n* of file *file-spec*
Invalid table nesting. Ignored.

Warning: You coded a nested table incorrectly. There are three possible problems:

1 You specified a <TABLE> tag with a caption argument in an argument to a <TABLE_ROW> tag. You cannot place a formal table inside another formal table.

2 You specified a <TABLE> tag inside another table, but outside of a <TABLE_ROW> tag. All text (including nested tables) that occur within a table must be coded within <TABLE_ROW> tags.

3 You specified <TABLE_UNIT> inside a table that is nested in another table. <TABLE_UNIT> tags are not valid in nested tables.

User Action: Based on the above situations, you should take one of the following actions:

1 Remove the caption argument from the nested table tag.

2 Nest the table inside a <TABLE_ROW> tag.

3 Remove the <TABLE_UNIT> tags.

TBLNOROWS, at tag or text on line *n* of file *file-spec*
A table was specified without any table rows.

Error: <TABLE> and <ENDTABLE> tags were encountered in the file, but there were no <TABLE_ROW> tags between them.

User Action: Verify that you specified <TABLE_ROW> tags correctly. Or, remove the <TABLE> and <ENDTABLE> tags if you did not want to place a table there.

TBLNOTDEC, at tag or text on line *n* of file *file-spec*
Table footnote *number* is referenced but not declared.

Warning: A <FOOTREF> tag is specified in a table, but there is no corresponding <FOOTNOTE> tag at the beginning of the table. The footnote reference will not be printed.

User Action: Verify that you specified the <FOOTNOTE> tag correctly at the beginning of the table. Verify that the number specified in the declaration matches the number in the <FOOTREF> tag.

TBLNOTROW, at tag or text on line *n* of file *file-spec*
Tag <*tagname*> specified outside of <TABLE_ROW> .

Error: A tag is specified between two <TABLE_ROW> tags and is therefore not valid in this context.

User Action: Correct your source file to move the tags and related text, if any, to within the bounds of a <TABLE_ROW> tag.

TBLNOTSET, at tag or text on line *n* of file *file-spec*
Table from line *number* has no <TABLE_SETUP> attributes.

Error: A <TABLE_ROW> tag was encountered in a table without a <TABLE_SETUP> tag.

User Action: Correct your source file to specify the number of columns and widths for the table.

TBLNOTMTP, at tag or text on line *n* of file *file-spec*
<*tagname*> invalid in table that is not multipage

Error: A tag, for example <TABLE_UNIT> was specified in a table that does not have the multipage attribute.

User Action: Determine whether the table is a multipage table. If so, be sure that it is not specified with the KEEP attribute.

TBLSPANNO, at tag or text on line *n* of file *file-spec*
Argument *number* to tag <SPAN> is invalid for the current column count.

Warning: A <SPAN> tag's argument exceeded the number of remaining columns in the table. For example, you specified <SPAN> (3) in a table with only 2 columns.

User Action: Correct your source file to specify the number of columns to span. Be sure that your numeric argument, plus the current column number, do not exceed the number of columns in the table.

TMPNONEST, at tag or text on line *n* of file
file-spec
Invalid attempt to nest reference templates.
Last template was *string*

Error: A tag invoking a reference template was encountered when a reference template was already in effect; for example, a <ROUTINE_SECTION> is specified before a previous <COMMAND_SECTION> was terminated.

User Action: Correct your source file to correctly terminate the reference section that was not terminated.

TMPTAGDEF, You cannot redefine *<tagname>* using *<tagname>* .

Informational: A tag that defines a template tag specified a tag name that is already a defined tag. This message indicates one of the following:

- Your input file has more than one reference template section, and you used the same template tag names in both. (For example, a <SET_TEMPLATE_ROUTINE> tag may specify the same tag name in more than one occurrence of <ROUTINE_SECTION> .
- You have specified a name that is the name of an SDML global tag.

User Action: If the message occurred because you used the same template tag names in more than one reference template section, you need take no action. If you have used a name that is the name of an SDML tag, you should select another name; future use of the SDML tag name in the same source file may produce unpredictable results.

TMPTAGEND, at tag or text on line *n* of file
file-spec
<tagname> not terminated before *<tagname>* .

Warning: A tag occurring within the context of a reference template was not terminated before the end of the file or before the beginning of the next reference section. For example, an <OVERVIEW> tag from a <COMMAND> description was not terminated before the next <COMMAND> description was encountered.

User Action: Correct the source file so that you correctly terminate the template tag.

TMPTAGNAM, at tag or text on line *n* of file
file-spec
string is an invalid template tag name.

Error: A template tag, for example <SET_TEMPLATE_COMMAND> , specifies that the tag name is already defined in SDML, or it is an invalid name, that is, it contains more than 31 characters or it contains nonalphanumeric characters.

User Action: Choose another name for the tag, and modify your source file.

UNBALPARS, at tag or text on line *n* of file
file-spec
Unbalanced parentheses in argument *number*

Warning: One or more left parentheses are present without matching right parentheses. A sufficient number of right parentheses are appended to the argument to achieve a balance.

User Action: Correct the unbalance, either by adding the proper number of right parentheses or by coding the left parentheses using the <OPAREN> tag.

UNDEFSYMS, There were *number* undefined symbol references

Informational: This message reports the total number of references that were made to undefined symbols. Only the first reference to a specific undefined symbol is reported.

User Action: Either define the undefined symbol, or correct the spelling of the symbols being referenced. The undefined symbols were reported in other messages.

UNDEFTAGS, There were *number* undefined tag invocations

Informational: This message reports the total number of invocations of undefined tags. Only the first invocation of a specific undefined tag is reported.

User Action: Either define the undefined tags, or correct the spelling of the tag being invoked. The undefined tags were reported in other messages.

UPDATINFO, Update page label *string*, *<tagname>*

Informational: A document that is being processed to produce update pages may have associated update information. The information indicated will be printed on the bottom of each page output.

User Action: Verify that the update page label is correct for your document.

USER_IMSG, *string*
Line is *number* of file *filename*

Informational: This message code is used for messages written to the terminal or output stream when a <USER_I_MESSAGE> tag is specified in the input file.

User Action: The action you take depends on the content of the user's message.

USER_WMSG, at tag or text on line *n* of file
file-spec
string

Warning: This message code is used for messages written to the terminal or output stream when a <USER_W_MESSAGE> tag is specified in the input file.

User Action: The action you take depends on the content of the user's message.

USETAGDEF, Using tag's default: *number*

Informational: This informational message accompanies other warning messages. It generally indicates that a default value is being taken.

User Action: Edit the SDML source file to correct the argument.

VBARINARG, at tag or text on line *n* of file *file-spec*
A vertical bar was found in argument *number* to tag <*tagname*> . Use the <VBAR> tag to code a vertical bar in an argument

Fatal: The tag translator treats the vertical bar character as a special operator character in arguments to tags. If you include a vertical bar character in an argument, the tag translator may overlook the right parenthesis that terminates the argument list, and may scan all of the remaining input looking for the proper terminator.

If you have mistakenly omitted the closing parenthesis of an argument list, the tag translator begins to search the remainder of your file for a closing parenthesis. If it encounters a vertical bar (perhaps in a figure or code example), it then begins to search for a matching ampersand. Thus, this error message may be issued instead of the EOFARGLST error message.

User Action: Correct the argument list by replacing the vertical bar with the <VBAR> tag, or by properly closing the argument list.

VMOVRFLOW, at tag or text on line *n* of file *file-spec*
Internal error. Failure to allocate additional space in virtual memory

Fatal: This error message is generally caused by one of two types of error in the SDML file. These errors, and the action that you should take to correct them are as follows

- An unterminated argument list. If you fail to terminate an argument list, the tag translator regards the text that follows the argument list as part of the last argument. Suppose you write

  ```
  <EMPHASIS>(always\BOLD> press the RETURN key.
  ```

  instead of

  ```
  <EMPHASIS>(always\BOLD) press the RETURN key.
  ```

  The tag translator does not realize that the right angle bracket should have been a right parenthesis. It tries to include all the following words as part of the second argument of the <EMPHASIS> tag. This means that it may store large portions of the SDML file in memory while it searches for a closing parenthesis to the argument list. If memory limits are exceeded, the VMOVRFLOW error message is issued.

- Too many tag definitions. The tag translator has no limit on the number of tag definitions. However, a limit is imposed by the memory allocation algorithm. When this limit is exceeded, it usually indicates that one or both of the following conditions are present:

  - A file that contains tag definitions is being repeatedly included. This file is often a global definitions file that is being used to define tags or symbols for text substitution. You can avoid memory limitation problems by using the <CHECK_FOR_INCLUSION> tag at the start of

such a file. This ensures that the file's definitions are actually loaded only once, even though the file is included many times.

- A tag's definition changes frequently during execution, perhaps when the tag is being used to temporarily hold a text string. If the tag's definition is established each time by the <DEFINE> tag rather than by the <REDEFINE> tag, the definitions pile up in memory. A reference to the tag always retrieves the latest definition, and if you never use the <UNDEFINE> tag to erase the latest definition, you will never see the old definitions, but they are still occupying memory space.

User Action: Determine if the error was caused by an unclosed argument list. The error is generally located by the information supplied in the error message.

Otherwise, examine your tag definitions to determine whether you are defining the same tag name multiple times. You can avoid memory limitation problems by defining your tags with the <REDEFINE> tag, rather than the <DEFINE> tag. Unless you specifically intend to "stack" tag definitions, and use the <UNDEFINE> tag to "pop" definitions off the stack, you should always establish a tag definition with the <REDEFINE> tag.

XRFINCMPL, at tag or text on line *n* of file
file-spec
The file containing cross reference symbols is incomplete.
filename

Fatal: During the reading of the file containing cross reference symbols, an error was detected that indicates that the file was not recorded correctly.

User Action: Submit a Software Performance Report.

XRFINVALD, at tag or text on line *n* of file
file-spec
The file containing cross reference symbols is invalid.
filename

Fatal: The file was opened without error, but it does not contain cross reference symbols.

User Action: The wrong file is being referenced. Check the file name supplied with the /PROFILE qualifier.

XRFNORENM, at tag or text on line *n* of file
file-spec
The temporary cross reference file cannot be renamed to
filename

Fatal: The tag translator creates a temporary cross reference file with a file type TMP. When the file is complete, it renames it with a file type XREF. The rename operation has failed.

User Action: Submit a Software Performance Report.

C.3 Text Processor Messages

ATPT , Font was read at *number* pt

Informational: This is an internal error.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADALPHA , Improper alphabetic constant

Warning: Text formatter expected to scan a constant consisting only of letters.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADCHAR , Text line contains an invalid character

Warning: Text formatter could not scan a character from current input line.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADCODE , Bad code number: *value*, must be between *smaller number* and *larger number*
using zero instead

Warning: Text formatter expected to find code value within range shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADCS , Bad control sequence: `*Command*'

Warning: Text formatter could not understand how user meant to use the shown command in the current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADDISC , Improper discretionary list. Must contain only
`kerns' and `boxes'

Warning: Discretionary list contained unexpected information.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADDISCMATH , Illegal `discretionary break' in `math' mode.
The third part of a discretionary break must be empty in this mode. Attempting to recover...

Warning: Discretionary break was improperly coded.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADDUMP , Cannot `\dump' inside a group

Warning: Command `\dump' occurred when there was one group or more still active.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADFONTFMT , Fatal font format file error

Error: Text formatter could not load the font format file, or the font format file was fatally flawed.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADFRACTION , Ambiguous fraction ignored.

Warning: Text formatter could not determine the meaning of a fraction.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADGLUE , Infinite glue. Attempting to recover

Warning: Text formatter was asked to typeset a box using infinite glue. Glue was made finite and text formatter attempted to continue.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADHALIGN , Improper `\halign'

Warning: Command `\halign' was invalid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADHRULE , Cannot use `\hrule' in an `\hbox'.
Use `\leaders' or `\hrulefill' instead

Warning: Command `\hrule' is invalid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADHYPHEN , Improper \hypenation ignored.
Must contain only letters and hyphens

Warning: Text formatter can only read letters and hyphens in a hyphenation pattern.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADHYPHFMT , Fatal hyphenation format file error

Error: Text formatter could not load the hyphenation format file, or the hyphenation format file was fatally flawed.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADLASTBOX , `\box255' is not empty, attempting to continue

Warning: Text formatter expected `\box255' to be empty.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADLEADERS , `\leaders' not followed by proper `glue'

Warning: Glue specification was not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADLETINPAT , Non-letter found in `\patterns'

Warning: Text formatter can only read letters in a `pattern'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADLIMIT , Ignoring misplaced `\limits' or `\nolimits' command

Warning: Text formatter did not expect to see `\limits' or `\nolimits' in this context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADMACROFMT , Fatal macro format file error

Error: Text formatter could not load the macro format file, or the macro format file was fatally flawed.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADMAG , Illegal magnification ratio: *bad number,* value set to *good number*

Warning: Text formatter could not set magnification to requested value.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADMATHCHAR , \`*font*' number *font-number* is missing this character: \`*character*'

Warning: Text formatter could not typeset the character shown, because that character was missing from the font shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADMATHDELIM , Inserted missing or misplaced math delimiter

Warning: Text formatter became confused while closing a math formula.

User Action: Check the SDML file for <MATH> sequences and verify that all parentheses are matched and that all math tags and delimiters are correctly terminated. If this does not correct the problem and you are under a service contract with Digital, call your customer service center. Otherwise, submit a Software Performance Report.

BADNOALIGN , \`\noalign' can only appear after \`\cr'

Warning: Command \`\noalign' is not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADOMIT , \`\omit' can only appear after \`\cr' or \`&'

Warning: Command \`\omit' is not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADOUTLOOP, Output loop repeated *number of cycles* times without doing a \`\shipout'
Attempting to recover. Increase \`\maxdeadcycles' if neccesary

Warning: Text formatter cycled through its output loop too many times without doing a \`\shipout'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPARAMNUM , Illegal parameter number

Warning: Text formatter became confused while scanning parameters to a macro.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPARAMSEQ , Parameter was numbered out of sequence

Warning: Text formatter became confused while scanning parameters to a macro because a parameter was numbered out of sequence.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPATTERNS , Bad \patterns

Warning: Text formatter became confused while reading a `pattern'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPREAMBLE , Cannot nest alignment preambles

Error: Text formatter became confused during current alignment, because it found an alignment within an alignment.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPREFIX2 , Can't use `\long' or `\outer' with `*command*'

Warning: Text formatter will ignore `\long' or `\outer' or `\global' when used with the shown command.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPREFIX3 , Can't use `\long' or `\outer' or `\global' with `*command*'

Warning: Text formatter will ignore `\long' or `\outer' when used with the shown command.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPREVGRAF , `\prevgraf' = *number* . Value must be positive

Warning: Text formatter expected to see `\prevgraf' specified with a positive value.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADPT , Bad font size = *number* pt, replaced by 10 pt

Warning: Text formatter attempted to read a font at an improper font size.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADUNBOX , Cannot `unbox' a *`hbox or vbox'* in *`Text formatter mode'* mode

Warning: command `unbox' is invalid under current circumstances.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADUNIT , Bad unit of measure, replaced by *`string'*

Warning: Text formatter could not use the unit of measure user asked for. Unit was changed to shown value.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADVALUE , *`quantity' = number* . Value must be between *smaller number* and *larger number*

Warning: Text formatter was expecting to read a value within the range shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BUFFERSIZE , Maximum number of characters in the input buffer is *number*

Informational: User has exceeded total size of text formatter's input buffer.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

CANNOTRESETMAG , Cannot change magnification, using old value: *number*

Warning: Text formatter can only set magnification value once.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

CONFUSION , Text formatter became confused doing this: *`Command'* , job is aborting

Error: Text formatter became confused doing the shown activity, and had to exit.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

CONFUSION , Text formatter became confused doing this: *`Command'* , job is aborting

Error: Text formatter became confused doing the shown activity, and had to exit.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DIMENTOOBIG , Dimension too large

Warning: Text formatter became confused while scanning a dimension, because the dimension was too big.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DISCTOOLONG , Discretionary list is too long. Attempting to recover

Warning: Discretionary list was too long.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DOUBLESUB , Double subscript

Warning: Text formatter found a quantity with two subscripts. In these cases, x^1^2 will be treated like x^1{}^2.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DOUBLESUPER , Double superscript

Warning: Text formatter found a quantity with two superscripts. In these cases, x_1_2 will be treated like x_1{}_2.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DURING , Error occurred during a *`command'*

Warning: Error occurred during a specified command or control sequence.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

DURINGOR , Error occurred during a *`command'* or *`command'*

Warning: Error occurred during one of the two commands or control sequences shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

EMERGENCYSTOP , Job is aborting

Error: Text formatter became hopelessly confused, and must exit.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ENDDURING , `\end' occurred during *`command'*

Warning: The command `\end' occurred unexpectedly.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ENDINGROUP , `\end' occurred inside group at level *number*

Warning: The command `\end' occurred when there was one group or more still active. Examine the input and output files and attempt to determine what tag was not terminated.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

EOF , Unexpected end of file

Warning: Text formatter encountered the end of the current file unexpectedly.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

EOP , Paragraph ended before the following completed:

Warning: Text formatter encountered the end of a paragraph unexpectedly.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

EXTRATAB , Extra `\halign' or `&' in alignment has been changed to `\cr'

Warning: Text formatter found extra or unexpected `\halign' or `&'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

FILENAME , *`file-name'*

Informational: Identifies current file being used as input.

User Action: This message always occurs in conjunction with other messages. Action depends upon associated message.

FONTDIMEN , The following font has only *number* fontdimen parameters
To increase the number of font parameters you must use
\fontdimen immediately after the \font is loaded

Warning: The current font has too many parameters.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

FONTERROR , Error occurred in the following font:

Warning: Text formatter found an error in the current font.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

FONTMEMSIZE , Font memory size = *number* words

Informational: Reports the total size of text formatter's font memory.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

FONTTYPE , Font was declared by: `*string*'

Warning: Identifies the font's type, e.g. \preloaded.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

FUNNYCS , Missing control sequence inserted

Warning: Text formatter became confused while defining a control sequence.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

IGNORED , Extra `*command*' ignored

Warning: An unneccesary text formatting command was found.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

IGNOREDTEXT , All text was ignored after line *string*

Warning: Text formatter could not typeset any of the text following the line shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

IGNORING , Ignored the following: `*command*'

Warning: Text formatter encountered an unexpected command, and ignored it.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

IMPROPERCMD , Improper command found: *`command'*

Warning: Command shown is not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INCOMPLETECMD , Incomplete command found: *`command'*

Warning: Text formatter became confused while trying to scan the shown command.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INFO , *text* - on page *number*

Informational: A text formatting macro has generated an informational message. A problem occurred while the text formatter was processing the output. *Text* provides a description of the problem. In most cases, the file will continue processing successfully and you can examine the output. The message indicates the page on which the problem occurred, and you can use this information to correct the problem.

User Action: The following is a list of the messages you may see and the associated action that will correct the problem.

- Callout number 0 undefined. Using 1.

  A <CALLOUT> or <CO> tag was specified as 0.

- Callout number exceeds 99. Using 99.

  A <CALLOUT> or <CO> tag's value is greater than 99. POSTSCRIPT output devices cannot print numbers greater than 99.

- Error in setting note type; using default.

  A local or personal DESIGN file specified an invalid value for the \noteformattype. Refer to the *VAX DOCUMENT Doctype Designer's Guide* for value values.

- Error. Maximum number of table footnotes exceeded.

  Too many <FOOTNOTE> tags were specified for a table. The largest number that is allowed is 12.

- Error pushing/popping revision bars.

  <MARK> and <ENDMARK> tags are not evenly matched, or are not suppressed around heading levels in the correct sequence. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

- Possible problem in output of revision bars.

  The text formatter encountered a complex situation while a revision mark (from the <MARK> tag) was active. Examine your output closely to determine if the revision mark output is correct. A revision mark may not appear, or appear where you did not intend it to. If there is a problem, you can change your source file to decrease the range in which a revision mark is active. For example, if you have changes in two rows of a table, you might place the <MARK> and <ENDMARK> tags inside each of the two <TABLE_ROW> tags instead of around the entire table.

- Figure is too large for a single page.

  A value specified in <FIGURE_SPACE> or <FIGURE_FILE> exceeds the depth of the output page. The text formatter will use only the depth of the page.

  Correct the invalid depth argument in the SDML file.

- <FOOTREF> before <FOOTNOTE> in table ignored.

  A tag <FOOTREF> tag occurred in a table for which no corresponding <FOOTNOTE> tag occured. Footnotes in tables must be specified at the beginning of the table, before any references to them occur.

- Graphics files cannot be included on this device.

  A <FIGURE_FILE> or <ICON_FILE> tag was encountered that specified a graphics file to be included on a device that does not support the inclusion of graphics. The tag is ignored.

- Index subentry more than 4 levels deep; ignored.

  An indexing tag (<X> or <Y>) contained more than four <XSUBENTRY> tags. Only four levels of subentry are allowed.

  Correct the indexing tag to remove the excessive subentries.

- Included files are not allowed to float.

  An included text formatter file in a figure is assumed to be multipage and cannot float.

  Remove the FLOAT attribute.

- Internal error while processing a table.

  The text formatter has received bad input. Submit an SPR.

- Invalid table of contents entry.

  A local or personal doctype specified an invalid value for a table of contents entry. Correct the table of contents entry to use a valid code.

  If this error occurs using a standard VAX DOCUMENT design, submit an SPR.

- Monospaced example is too big to keep.

  The keyword KEEP was specified as an attribute to a <CODE_EXAMPLE> tag or other tag that specifies a monospaced example. The number of lines in the example cannot fit on a single page of output.

  Remove the keyword KEEP from the tag and provide <VALID_BREAK> tags to indicate acceptable page break points.

- Multipage figure will not be allowed to float.

  A figure was specified with both the attributes FLOAT and MULTIPAGE. Multipage figures cannot float; the text processor will output it at the location of its occurrence in the source file.

  Remove the FLOAT attribute.

- No template heading defined at this level; using level two.

- <PAGE> ignored in floating figure.

  A <PAGE> or <FINAL_CLEANUP> (PAGE) tag was encountered in a figure that had the default attribute float.

  Specify <FIGURE_ATTRIBUTES> (MULTIPAGE) for all figures that are more than a page in length.

- <PAGE> tag ignored in table that is not multipage.

  A <PAGE> or <FINAL_CLEANUP> (PAGE) tag was encountered in a table that had the default attribute float.

  Specify <TABLE_ATTRIBUTES> (MULTIPAGE) for all tables that are more than a page in length.

- Possible table paging problem.

  A table that is output on more than one page is being forced to break across pages without explicit page breaking or sufficient valid table break points.

  In most cases, the output will be perfectly all right. The text formatter makes its best decision about breaking tables across pages when there are <TABLE_ROW_BREAK> tags to specify the first and last valid breaks for the table.

- Revision bars cannot be output on this device.

  <REVISION>, <MARK>, and <ENDMARK> tags cannot be processed for this device. No change bars will be printed.

- <SPAN> at end of table column ignored.

  The <SPAN> tag, which specifies that text is to span 2 or more columns in a table, must *precede* the text in the argument. For example,

  ```
  <TABLE_ROW>(<SPAN>(3)This text sits across all three columns)
  ```

  Move the <SPAN> tag to the beginning of the tag's argument.

- The table, as specified, is too wide for the page.

  The values specified in <TABLE_SETUP> result in a table that will not fit on the current page. Recheck the arguments you specified to the <TABLE_SETUP> tag, and decrease them. You will be able to tell from the output how much of the table is excessive.

- <TABLE_HEADS> preceding <TABLE_SETUP> ignored.

  Table headings must be specified following the <TABLE_SETUP> tag. Move the <TABLE_HEADS> so that it precedes the <TABLE_SETUP> tag.

- <TABLE_HEADS> within <TABLE_UNIT> ignored.

  The <TABLE_HEADS> should not be used inside <TABLE_UNIT> sequences; use the <TABLE_UNIT_HEADS> tag instead.

- <TABLE_ROW> encountered outside of a <TABLE> .

  A <TABLE_ROW> tag was specified, but there was no <TABLE> tag. The tag will be ignored. Correct the file so that a valid <TABLE> tag precedes it.

- <TABLE_ROW_BREAK> tags in single-page table ignored.

  The tag <TABLE_ROW_BREAK> was specified in a table that had the attribute KEEP specified.

  Remove the KEEP attribute from the table or remove the <TABLE_ROW_BREAK> tag.

- <TABLE_UNIT_HEADS> outside of table unit ignored.

  A <TABLE_UNIT_HEADS> tag was encountered before a <TABLE_UNIT> tag was specified to begin a table unit.

  Remove the tag; or insert a <TABLE_UNIT> tag to enclose a table unit.

- Tracing is disabled in VAX DOCUMENT.

  An instruction to accumulate tracing information occurred in a TEX file. VAX DOCUMENT does not support these instructions.

- Too many columns specified for multi-column output.

  A local or personal doctype specified too many columns of output.

  If this error occurs using a standard doctype, submit an SPR.

INITEXPAT , `\patterns' can only be used by Initex

Warning: Text formatter only recognizes `\patterns' when it is building format files, not during a run of VAX DOCUMENT.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSERTED , Missing `*command*' inserted

Warning: A missing command was inserted into input stream.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSERTHBOX , `\insert' can only be added to a `\vbox'

Warning: command `\insert' is not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSERTING , Inserting *`command'*

Warning: Text formatter attempted to resolve an unexpected situation by inserting shown command into input stream.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSUFEXTFON , Math formula deleted: Insufficient `extension' fonts

Warning: Math formula could not be typeset because `extension' fonts were inadequate or missing.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSUFSYMFON , Math formula deleted: Insufficient `symbol' fonts

Warning: Math formula could not be typeset because `symbol' fonts were inadequate or missing.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INTERNALERROR , Internal error number: *`error number'*. Job is aborting

Error: Text formatter found an internal inconsistency.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

LINE , Error occurred on or around line number: *number*

Informational: Reports the current line of input that the text formatter is processing.

User Action: This message always occurs in conjuction with other messages. Action depends upon associated message.

LINETOOLONG , Line too long by *Integer.remainder* points

Informational: The text formatter found an overly lengthy line.

User Action: This message always occurs in conjuction with other messages. Action depends upon associated message.

LINETOOSHORT , Line too short - on page *Page number*

Informational: The text formatter found a line that was too short.

User Action: This message always occurs in conjuction with other messages. Action depends upon associated message.

MATHOVERFLOW , Arithmetic overflow

Warning: The result of a mathematical operation was too large.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXFONTS , Maximum number of fonts is: *number*

Informational: User has exceeded maximum number of fonts.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXGROUPS , Maximum number of groups is: *number*

Informational: User has exceeded maximum number of groups.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXHASH , Maximum number of entries in hash table is: *number*

Informational: User has exceeded maximum number of entries on text formatter's hash table.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXHYPHS , Number of hyphenation exceptions is: *number*

Informational: User has exceeded maximum number of hyphenations.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXINPUTS , Maximum number of simultaneous input files is: *number*

Informational: User has exceeded maximum number of simultaneous input files.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXNESTS , Maximum number of simultaneous semantic levels is *number*

Informational: User has exceeded maximum number of simultaneous semantic levels in text formatter.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXPARAMS , Maximum number of simultaneous text macro parameters is *number*

Informational: User has exceeded maximum number of parameters to a macro.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXSAVE , Maximum number of entries on save stack is: *number*

Informational: User has exceeded maximum number of entries on text formatter's save stack.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MAXSTRINGS , Maximum number of strings is: *number*

Informational: User has exceeded maximum number of strings.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MEMORYSIZE , Main memory size = *number* words

Informational: User has exceeded total size of text formatter's main memory.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MESSAGE , *text* - on page *number*

Warning: A text formatter macro generated a warning message. The following is a list of possible messages and actions.

User Action: See the action associated with each of the following messages.

- Error in window size.

  An error occurred in a local or personal doctype; an invalid value was specified in a \window.

  If this error occurs in a standard doctype, submit an SPR.

- Internal error in setting block indents.

  Too many arguments were specified to \blockindents, or the values were not valid. This error should only occur if a local or personal DESIGN file contains an invalid value.

- Internal error in figure key definition.

  Submit an SPR.

- Too many floating figures.

  There are too many <FIGURE> tag sequences within a short span of text specified, and the default attribute "float" is in effect for all of them.

  The text formatter cannot produce meaningful output when there are too many figures that are floating. When you have a large number of figures in a short span of text, you should use the <FIGURE_ATTRIBUTES> (KEEP) tag to keep them in sequential order.

- No room for a new [item].

  Internal use.

MISSINGBOOL , `<' or `=' or `> '

Warning: Text formatter expected to scan one of the Boolean values shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report. Determine which Boolean value is needed and insert it in the right place.

MISSINGLBRACE , Possible missing `{'

Warning: Text formatter did not see an `{'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MISSINGRBRACE , Possible missing `}'

Warning: Text formatter did not see an `}'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

MODE , Error occurred in `*string*' mode'

Informational: Reports the current mode that the text formatter is in.

User Action: This message always occurs in conjunction with other messages. Action depends upon associated message.

MUERROR , Mismatched glue units. Assuming that 1mu=1pt

Warning: Text formatter scanned glue specifications that are incompatible.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOBOX , Expected to find `\hbox' or `\vbox' or `\copy' or `\box' or something like that. Some text may be missing from output.

Warning: Text formatter expected to find a box.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOEND , No `\end' was found

Error: Text formatter expected to see a `\end' command.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOFONTID , Could not find font identifier for the following font:

Warning: Font identifier for current font is missing.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOINSLASTBOX , `\insert255' is illegal. Changing to `\insert0'

Warning: Text formatter register `\insert255' cannot be used in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOMATHACCENT , Changed `\accent' to `\mathaccent'

Warning: The text formatter command `\accent' works differently in formulas than it does in normal text.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NONUMBER , Missing number, using zero

Warning: Text formatter expected to find a number.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOROOM , Exceeded memory capacity:

Error: Text formatter exceeded one of its memory capacities. The message immediately following indicates which quantity was exceeded.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOUSEAFTER ,   After this: *'command.'*, the following is invalid:

Warning: Text formatter cannot use the command shown in the next message directly after the command shown in this message.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NUMBERTOOBIG ,   Number too large, using 2147483647 instead

Warning: Text formatter could not use specified number because it is too big. Text formatter used 2147483647 instead, which is the largest number possible in that context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ONEOFFOUR ,   Some or all of the following 4 messages apply:

Warning: One or more of the four messages that follow on your screen apply.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ONEOFTWO ,   One or both of the following messages apply:

Warning: One or both of the two messages that follow on your screen apply.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ONPAGE ,   on page *number*

Informational: Identifies the current page being output.

User Action: This message always occurs in conjunction with other messages. Action depends upon associated message.

OUTFILENAME ,   *'file-name'*

Informational: Identifies the output file name.

User Action: None.

PAGESOUT ,   *number* page(s) written.

Informational: Reports the number of pages written to the output file.

User Action: None.

PAGETOOBIG ,   Cannot ship out huge page. More than 18 feet wide or long.

Warning: Text formatter became confused during typesetting of current page because page was too large.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

PAGETOOLONG , Page too long by *string.string* points

Warning: Text formatter could not fit contents of current page into boundaries of current page. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

PAGETOOSHORT , Page too short - on page *Page number*

Informational: The text formatter had trouble composing the current page. There were not enough lines to reach the bottom. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

PATTERNEXISTS , Duplicate `pattern' found

Warning: Text formatter found a duplicate `pattern'.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

PATTERNMEMSIZE , Hyphenation pattern memory size = *number* words

Informational: User has exceeded total size of text formatter's hyphenation memory.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

POOLSIZE , String memory size = *number* bytes

Informational: User has exceeded total size of text formatter's string memory.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

RUNAWAY , Runaway *`kind of text'*

Warning: Text formatter became confused while scanning the shown kind of text.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

SCALED , Font was scaled: *number*

Informational: This is an internal error.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

SHOWCONTEXT , *'text'*

Informational: Displays input stream that the text formatter was attempting to read when something went wrong.

User Action: This message always occurs in conjunction with other messages. Action depends upon associated message.

SHOWTOKEN , *'text'*

Informational: Displays part of text formatter's current token list.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

TIGHTSPACING , Insufficient inter-word spacing - on page *Page number*

Informational: The text formatter had trouble composing a line - there was not enough white space on the line. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

TOOFEWLINES , Excess inter-line spacing - on page *Page number*

Informational: The text formatter had trouble composing the current page. There was too much white space between the lines. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

TOOMANYERRORS , Error limit is 30

Error: Text formatter found more than 30 errors.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

TOOMANYLINES , Insufficient inter-word spacing - on page *Page number*

Informational: The text formatter had trouble composing the current page. There was not enough white space between the lines. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

TOOMANYPARAMS , More than nine parameters were passed to a macro

Warning: Text formatter became confused while scanning parameters to a macro because there were too many parameters.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

TRACE , *text*

Informational: User has turned on text formatter's tracing mechanism. Text contains information about VAX DOCUMENT's text formatter's internals.

User Action: None.

UNDEFINEDCS , Undefined control sequence

Warning: Text formatter encountered an undefined control sequence.

User Action: This can happen if a TEX file, created using one doctype, is reprocessed using another doctype; for example, if you created a TEX file using the LETTER doctype then reprocessed the file using /NOTAG and the OVERHEADS doctype. Reprocess the file using /TAG_TRANSLATORT and /TEXT_FORMATTER. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

UNKNOWNUNIT , Dimension unit must be: em,ex,in,pt,pc,cm,mm,dd,cc,bp,sp

Warning: Text formatter expected to read one of the dimensions shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

USINGZERO , Bad `*value*', must be between *smaller number* and *larger number*, using zero instead

Warning: Text formatter expected to scan a value in the range shown.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

VSPLITHBOX , Attempted to `\vsplit' a `\hbox'

Warning: Command `\vsplit' is not valid in current context.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

WIDESPACING , Excess inter-word spacing - on page *Page number*

Informational: The text formatter had trouble composing a line; there was too much white space on the line. Such errors are common with a line printer because it is monospaced.

User Action: Examine output on page shown to determine if there is a problem with the coding of the SDML file. Ignore the problem if it is not serious. You also have the options of changing destinations or changing the doctype, if possible.

C.4 Device Converter Messages

BADFILABORT , file bad—aborting run

Error: The device converter cannot use the font file named in an adjacent message.

User Action: Another file may have been given the same name. Use the DIRECTORY/FULL DCL command to see that the size and format are similar to other font files. If necessary, reinstall the font data.

BADFONT , font file bad—using blanks

Warning: The file that describes the font design is unusable. The program will substitute blanks for the characters in this font.

User Action: Use the DIRECTORY/FULL DCL command to see if the size and format of this file resembles other font files. If necessary, reinstall the font data.

BADINCLUDE , included file is bad

Warning: The program has encountered errors in processing the contents or index file requested.

User Action: Check that previous processing steps generated the contents or index file expected; this may be an old, incompatible file, it may be another file with the same name, or it may be the correct file made unreadable because of an I/O error.

BADMETRIC , bad metric file

Error: The device converter has encountered errors in processing the indicated font metric file.

User Action: This may be another file named the same; check the file size and format with the DIRECTORY/FULL DCL command to see if it resembles other font metric files. It may be necessary to reinstall the font data.

CANNOTCONNECT , cannot connect along a diagonal

Warning: While attempting to draw change bars, the program has encountered an attempt to draw a non-vertical line. Only vertical change bars are allowed.

User Action: This is controlled by the macros that describe change bar positioning. Look there for errors and inconsistencies.

CANNOTCREATE , cannot create output file

Error: The program cannot create an output file.

User Action: Check that the output directory exists, that the user has permission to write in it, and that the device is available and is not full.

CANNOTOPEN , cannot open input file

Error: The program cannot open the input DVI file.

User Action: Check that the input directory exists; check that the file exists or was created by earlier processing steps; check that the user has permission to read the file, and that the device is available.

CONFIGERROR , error in configuration file

Error: An error has been detected in reading the file that describes POSTSCRIPT fonts. An adjacent error message gives more details.

User Action: Check that the device is available and that the user has permission to read the file. Take the action suggested by the adjacent error message. If necessary, reinstall the font data.

FILENESTING , one included file cannot include another file

Warning: One contents or index file may not include another.

User Action: Check the macros that describe contents and index files to see that they are correctly generating the filespecs; see if the filespec generated for the contents or index file is the same as an existing file.

FONTCONFIG , font description missing or illegal in configuration file

Error: The specified font is not present, or not fully described, in the file that describes POSTSCRIPT fonts.

User Action: If necessary, reinstall the POSTSCRIPT support.

FONTERROR , error in font
string

Warning: An error has occurred while processing the named font. The specific problem is identified in an adjacent message.

User Action: Note the name of the font. Take the action suggested by the adjacent error message.

FONTTOOBIG , too many characters in font—limit is 188

Error: The document uses too many characters from the named font.

User Action: Reinstall the font files. If problem persists, and if you are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a software performance report (SPR).

IGNORING , ignoring included input file
filespec

Warning: The program cannot process the included DVI file for the reason specified in an adjacent message.

User Action: Note the name of the file. Take the action suggested by the adjacent error message.

INCLUDING , including input file:
filespec

Informational: Reports that the program is processing a DVI file other than the file named on the command line; this is typically a contents or index file.

User Action: None.

INPUTBAD , bad input file—aborting run

Error: The program has encountered errors in the named input file and cannot continue.

User Action: Check the messages generated by earlier processing steps to see if a complete DVI file was created.

INPUTFILE , input file is:
filespec

Informational: Reports the name of the primary DVI file; i.e., the file named on the command line.

User Action: None.

INTERNALERROR , internal error

Error: A problem internal to the device converter prevents it from continuing.

User Action: This sometimes represents a document so large or complex that it exceeds internal buffer sizes; sometimes you can work around this by running contents and index processing as separate jobs, rather than including them in place. You may be able to process the document in two or more pieces by using the STARTING_PAGE and ENDING-PAGE or NUMBER_OF_PAGES parameters to the device converter. If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INVALIDPOINT , invalid point number `*number*'

Warning: An error has occurred in the macros used to describe change bars. Specifically, an attempt has been made to locate a point off the page, or the point is incorrectly described.

User Action: This is controlled by the macros that describe change bar positioning. Look there for errors and inconsistencies.

NOFILABORT , cannot open file—aborting run

Error: The device converter cannot open the named file.

User Action: Check to see that the device is available, that the file exists, and that the user has privileges to access that file.

NOFONT , cannot open font file—using blanks

Warning: The device converter cannot open the named font file.

User Action: Check that the device exists and is available, and that the user has permission to read the file.

NOINCLUDE , cannot open included file

Warning: The device converter cannot open the contents or index file named in the adjacent message. Either the pathname and filename are incorrect, or the program cannot gain access because of file protection or locking by another process. Processing will continue without this file.

User Action: Check that the named file exists and that the user has access to it.

NOINPUTPAGE , no pages in input file

Error: The input file named in an adjacent message contains no pages; therefore, there will be no output to print.

User Action: Check to see if errors reported in earlier processes caused there to be no pages generated.

NOMETRIC , cannot open metric file

Error: The font metric file named in an adjacent message does not exist, or the user does not have access to it because of file protection or problems with the device.

User Action: Check that the file exists and is available, and that the user has access to it. Check that the user is specifying a DVI input file appropriate for the destination requested.

NOSIXELIMAGE , cannot find sixel image in figure file

Warning: The figure file named in an adjacent message does not contain the command sequence that identifies a sixel image. White space appears where the figure is expected.

User Action: Check the figure file. For LN03 use, it must be a sixel file.

NOSUBSTITUTE , no adequate alternate font found

Error: The text formatter has requested a down-loaded font for which the device converter cannot find a reasonable substitute.

User Action: Check the document design for the fonts used; check that the corresponding font files exist at the sizes desired.

NOSUCHUNIT , no such unit of measure

Warning: A DVI special command used to position change bars or describe graphics files contains an unknown unit of measure, or no unit at all.

User Action: This is controlled by the macros that describe change bar positioning or figure file characteristics. Look there for errors.

OFFPAGE , cannot set text outside page boundaries

Warning: An attempt has been made to set text outside the boundaries of the physical page specified.

User Action: Check the messages from the text formatter for lines too long; check that you specified a destination that can print a page as large as required by your design; check to see if you specified a horizontal or vertical offset that will push your design off the page.

ONPAGE , on page [*string*]

Informational: Reports the number of the output page on which the condition reported in the associated messages occurred.

User Action: Note this page number if you need to refer to the output.

OUTPUTFAIL , error writing to output file

Error: The program encountered an error in writing to the output file. This usually means that the output disk is full, the user's disk quota is exceeded, or the output disk has become unusable.

User Action: Check that the output device is usable. If the disk is full or the user's disk quota is used up, delete unwanted files and run the device converter again.

PAGENOTFOUND , starting page not found

Error: The user has specified a starting page number, but that page does not exist in the input file.

User Action: Check the starting page specified; check the input file specified.

PAGESOUT , *number* page(s) written to file:
filespec

Informational: This reports the number of pages written to the output file named.

User Action: No direct action needed; however, if the number is zero, or less than expected, check other messages for errors or examine the commands, filespecs, options, and input files for problems.

PAGETOOCOMPLEX , cannot hold all font data—using blanks

Warning: The amount of font data required to describe all the characters on this page will not fit in the LN03 font memory, so the characters in the current font will be represented as blanks.

User Action: If using a design that allows you to change text_size, choose a smaller size; or use only one kind of emphasis on the page if possible; or move portions of the text to adjacent pages.

PLOTFILEFAIL , cannot open figure file

Warning: The program cannot find the figure file named, or can find it but cannot open it.

User Action: Check that you have specified the right filespec, that the file exists in the path specified (or current default directory, if no path is specified), that you have permission to read that file, and that the file is not locked by the action of some other program running at the same time.

PROLOGERROR , error in prolog file

Error: The device converter cannot read its file that contains the POSTSCRIPT prolog for output.

User Action: Check that the device is available, that the file exists, and that the user has privileges to access the file. If necessary, reinstall the POSTSCRIPT support.

SPECIALERROR , error in \special
string

Warning: This lists the text of a DVI special command as received by the device converter. This command is generated by macros, but may contain some user-specified information, such as a filespec or size. The specific problem associated with this special will be described in an adjacent message.

User Action: Consult the adjacent message for a description of the problem. If necessary, check this message for a filespec or size that may have been incorrectly entered in a tag describing a figure file, contents file, or index file.

SPECTOOLONG , \special too long—will try to process anyway

Warning: This DVI special command exceeds the maximum length of 1000 characters. The program will truncate the command and try to process it.

User Action: Check the macros that generate this special command—they may be appending too many spaces or extraneous text following the command.

TOOMANYDVI , too many included input files

Warning: Too many included contents or index files have been requested.

User Action: Remove excess contents_file and index_file tags from input file.

TOOMANYFONTS , too many fonts—limit is 100

Error: The document contains too many fonts.

User Action: Change the document design so it does not require so many fonts, or use SDML tags that do not require so many fonts, or break the document into separate documents, or process the contents or index separately.

TOOMANYGLYPHS , too many glyphs used in all fonts combined

Error: The document contains too many characters in too many fonts. The LN03 cannot hold this many characters.

User Action: Break the device conversion of this document into two or more jobs, specifying the starting page and ending page or number of pages. End one job before the page on which this error was reported.

UNRECOGOPTN , unrecognized option

Warning: This DVI special command contains an option not supported by this device converter, or the program cannot find an option at all.

User Action: Check to see if you have requested a figure file, contents file, index file, change bars, or other function which is not supported for this destination; check for errors in the macros that generate this special command.

C.5 Index Facility Messages

ARGLST_TOO_LONG , The argument list is too long, or is not terminated.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

ATRLST_INVALID , The attribute list is not validly constructed.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADLOGIC , internal logic error detected

Error: The INDEX utility has detected an internal failure.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

BADVALUE , '*string*' is an invalid keyword value

Error: The indicated string is not a valid keyword value.

User Action: Reprocess the input file specifying a correct value for the qualifier keyword.

BEGIN_QUOTE , Needs a " character or a ` character to begin string.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

CANT_CREATE_BRN , Can't create intermediate file

Error: A necessary intermediate file cannot be created.

User Action: Check to be sure your INX file has not been corrupted, that you have sufficient disk space, that you have access to the current directory, and that there are no other factors in your current environment that would prevent the creation of a file during processing.

CREATED , '*string*' created

Success: The indicated output file has been created.

User Action: None.

CUT_PAGE_PREFIX , Page prefix was truncated to *string*.

Warning: A page prefix is greater then eight characters.

User Action: If truncation is not accepted, change the prefix to eight characters or less in the INX file and resubmit.

DUPBEGIN , duplicate BEGIN attribute ignored.

Warning: A BEGIN was encountered within the scope of another BEGIN.

User Action: Check the index entries in your source file to be sure that BEGIN's and END's match.

EMPTYIN , empty intermediate input file.

Warning: The intermediate work file is empty.

User Action: Check the command line to be sure that the input file was spelled correctly.

ENDPASS_1 , End of first pass over input file: *`filename'*

Informational: Indicates that the INDEX utility has started processing the specified input file.

User Action: None.

ENDPASS_2 , End of second pass over the input.

Success: The indexing utility has completed processing the input file.

User Action: None.

END_QUOTE_1 , Needs a ' character to end string.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

END_QUOTE_2 , Needs a " character to end string.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

IGNORED , *`string'* ignored

Warning: The indicated command line qualifier or qualifier keyword was ignored. One or more messages follow to indicate the reason.

User Action: See the explanation in the error messages that follow on your screen.

INPUT_OVERFLOW , Overflowed input put-back stack; element too large.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

INSVIRMEM , insufficient virtual memory

Error: The INDEX utility does not have sufficient virtual memory to generate the index.

User Action: Either reduce the size of the index or increase the SYSGEN parameter, VIRTUALPAGECNT, reboot, and reprocess your file.

INVINPUT , invalid input file format for intermediate file.

Warning: The intermediate input file is not compatible with the INDEX utility. It may have been corrupted.

User Action: Reprocess to get a new file.

INVRECORD , invalid record type in intermediate file.

Warning: The intermediate input file is not compatible with the INDEX utility. It may have been corrupted.

User Action: Reprocess to get a new file.

LANG_IGNORED , LANGUAGE tag not at beginning of file - being ignored.

Warning: The language tag was encountered in the INX file after index entry processing has started.

User Action: Reposition the language tag to the start of the INX file.

LANG_NAME , The *string* language name is not valid.

Fatal: The language specified is not supported.

User Action: Modify the language tag in the INX file to specify Danish, English, Finnish, French, German, Italian, Norwegian, Portuguese, Spanish, or Swedish.

NEEDS_START , Needs a = character in front of the string.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOBEGIN , END attribute ignored; no corresponding BEGIN.

Warning: An END was encountered and no previous BEGIN was encountered.

User Action: Check the index entries in your source file to be sure BEGIN's and END's match.

NOBRN , error converting `*string*' to internal file

Error: The indicated input file was not converted to an internal index workfile.

User Action: Correct any problems noted by the file conversion routine.

NOEND , BEGIN has no corresponding END

Warning: A BEGIN was encountered and the matching END was not.

User Action: Check the index entries in your source file to be sure BEGIN's and END's match.

NOINDEX , no index information in intermediate file.

Warning: No index information was found in the intermediate work file.

User Action: Check your input file to see if it includes any index entries.

NOLIST , parameter list not allowed

Warning: More than one file was specified on a line of a master index options file.

User Action: Modify the options file so that it contains only one name on each line.

NOREF , page reference not found

Error: An internal logic error was detected.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NOT_UNDELIM_STR , Not an undelimited string.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NO_ARGLST , *string* tag not followed by a starting argumentlist.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

NO_ENTRY_BLOCK , Attribute build failure - no entry block built.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

OPENIN , error opening *`filename'* for input

Error: The input file is erroneous, the file does not exist, or you do not have access to the file.

User Action: Check the file specification for errors and check to see if you have access to the file.

OPENOUT , error opening *`string'* for output

Error: An output file cannot be opened. This message is usually accompanied by a VAX RMS message indicating the reason for the failure.

User Action: Take corrective action based on the associated message.

PAGE_NUMBER , Illegal page number.

Error: An error occurred when the indexer read your INX file.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

SECTION_NAME , The *string* section name is not valid.

Error: The indicated section name is not CHAPTER, PREFIX or APPENDIX.

User Action: Change the section name in the INX file and resubmit.

SKIPPED , *n* reference(s) inside page range - ignored

Warning: The INDEX utility detected n index entries inside a BEGIN/END page range. These entries were ignored.

User Action: Check the input file to be sure that the BEGIN and END are positioned properly. If so, remove the intervening index entries.

SYNTAX , error parsing *`string'*

Warning: The indicated string encountered during the processing of a master index options file is erroneous. One or more messages follow indicating the nature of the error.

User Action: See the description in the messages that follow on your screen.

TEXT , *string*

Informational: This message follows other error messages that are generated when an error in a master index file or text processor character file is detected. It shows the full line of the options file that was in error. It is accompanied by one or more other error messages.

User Action: See the action for the accompanying error messages on your screen.

TEXTD , entry text: *`string'*

Informational: This message follows other messages generated due to some error in an index entry. It shows the complete text of the index entry.

User Action: See the action for the accompanying error messages on your screen.

TOKEN_OVERFLOW , A token is more than *string* characters long.

Error: A control string is more than the indicated length.

User Action: Check the item reported and modify the INX file for proper format.

TOODEEP , maximum subindex depth exceeded

Warning: An index entry containing ten or more levels of subindexing was detected. The subindexing was ignored.

User Action: Generate index entries with fewer levels of subindexing.

TRUNCATED , string too long - truncated

Warning: A character string is too large to fit into an INDEX utility internal buffer.

User Action: If you receive this message and are under a service contract with DIGITAL, call your customer service center. Otherwise, submit a Software Performance Report.

VALERR , specified value is out of legal range

Error: This message is part of a compound message.

User Action: Look at the associated messages for an indication of what is out of range.

WHERE , in *string* tag on line *number* of the INX file.

Informational: Locates the error.

User Action: See the action for the accompanying error messages on your screen.

WHERE , in *string* tag on line *number* of the INX file.

Informational: Locates the error.

User Action: See the action for the accompanying error messages on your screen.

D Summary of VAX DOCUMENT Tags

Table D-1 provides an alphabetic list of the tags that VAX DOCUMENT recognizes. For each tag, the doctype is given. The doctype is shown in the table as Global or as one of the doctypes.

If the doctype is specified as Global, the tag is recognized in all doctypes.[1] Global tags are described in *VAX DOCUMENT User Manual, Volume 1*.

If the doctype listed is one of the standard doctypes, the tag is recognized only for that doctype and is described in *VAX DOCUMENT User Manual, Volume 2*.

Table D–1 Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<ABSTRACT>	ARTICLE	Creates an article abstract and can also specify a heading for that abstract.
<ABSTRACT>	Global	Begins a summary description of a document on the title page or part page of a document.
<ACCENT>	Global	Supplies a freestanding accent mark.
<ACKNOWLEDGMENTS>	ARTICLE	Creates an acknowledgments section in an article.
<ALIGN_AFTER>	Global	Allows you to position formatted text in a list.
<ALIGN_CHAR>	Global	Identifies a nonprinting character to be used to align numeric information within a column of a list or table.
<ALIGN_NUMBER>	Global	Specifies a numeric value with alignment characters.
<AMPERSAND>	Global	Supplies an ampersand within an argument to a tag or in math.
<APPENDIX>	Global	Begins an appendix.
<ARGDEF>	SOFTWARE	Begins the text that defines an item in an argument definition list.
<ARGDEFLIST>	SOFTWARE	Begins a definition list describing zero or more routine arguments.
<ARGITEM>	SOFTWARE	Labels one to seven routine argument items to be defined in an argument definition list outside of the Routine template, or a single routine argument and its attributes within the Routine template.
<ARGTEXT>	SOFTWARE	Labels definition text in an argument definition list which replaces the information contained in a pair of <ARGITEM> and <ARGDEF> tags.
<ARGUMENT>	SOFTWARE	Emphasizes an argument name within text.
<AUTHOR>	ARTICLE	Specifies an author of an article.

[1] A few exceptions to this rule are noted in the introductions to the doctypes in *VAX DOCUMENT User Manual, Volume 2*.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<AUTHOR>	REPORT	Places the name of an author and one or two additional lines of information about the author in the front matter portion of a document.
<AUTHOR>	SOFTWARE	Places the name of an author and one or two additional lines of information about the author in the front matter portion of a document.
<AUTHOR_ADDR>	ARTICLE	Specifies the address of the author.
<AUTHOR_AFF>	ARTICLE	Specifies information about the organizational affiliation of the author.
<AUTHOR_INFO>	OVERHEADS	Specifies informational text about the overhead presentation.
<AUTHOR_LIST>	ARTICLE	Creates a list of authors for an article with multiple authors.
<AUTO_NUMBER>	OVERHEADS	Specifies that slides are to be numbered automatically, and that the slide number is to be placed at the bottom of every slide. Optionally, specifies a text string to be placed in front of the slide number on each page.
<BACKSLASH>	Global	Supplies a backslash within an argument to a tag.
<BACK_NOTE>	ARTICLE	Creates a back note entry, and creates a superscript reference number in the article text.
<BACK_NOTES>	ARTICLE	Causes any accumulated back notes to be output.
<BIBLIOGRAPHY>	ARTICLE	Begins a bibliography.
<BIB_ENTRY>	ARTICLE	Specifies a single entry in a bibliography.
<BOX>	Global	Produces a box that surrounds a user-specified character string.
<BYLINE>	REPORT	Places a name and other optional information below a ruled line in a signature list.
<BYLINE>	SOFTWARE	Places a name and other optional information below a ruled line in a signature list.
<CALLOUT>	Global	Labels a callout in an example. The <CALLOUT> tag is identical to the <CO> tag.
<CALLOUTS>	Global	Labels the beginning of a series of callouts contained in an example and enables the use of the <CALLOUT> and <CO> tags. (The <CALLOUT> tag is equivalent to the <CO> tag.)
<CALLOUT_REF>	Global	Labels a reference to a callout in text.
<CC>	LETTER	Lists the name of someone who is to receive a copy of a memo or letter.
<CCLIST>	LETTER	Begins a list of persons to whom you want to send a copy of a memo or letter.
<CENTER_LINE>	Global	Specifies a line of text that is to be centered in the current text margin.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<CHAPTER>	Global	Begins a chapter.
<CHEAD>	Global	Marks an unnumbered centered heading. Similar to the <CENTER> tag.
<CHEAD>	LETTER	Creates a centered heading.
<CHEAD>	REPORT	Creates a centered heading.
<CHECK_FOR_INCLUSION>	Global	Marks a file to ensure the file is included only once in the output.
<CLOSING>	LETTER	Specifies the closing of a letter.
<CO>	Global	Labels a callout in an example. The <CO> tag is identical to the <CALLOUT> tag.
<CODE_EXAMPLE>	Global	Begins an example of code. Code consists of words or lines of instructions written in a programming language or a command language.
<COLUMN>	ARTICLE	Specifies that a new column of output should begin in a two-column doctype.
<COLUMN>	REPORT	Specifies that a new column of output should begin in a two-column doctype.
<COMMAND>	SOFTWARE	Begins a new command description.
<COMMAND_SECTION>	SOFTWARE	Begins a command reference section, enables tags reserved for use in command sections, and sets paging attributes.
<COMMENT>	Global	Marks a portion of your SDML input file that you do not want to appear in your output. Text marked by a <COMMENT> tag is ignored by the tag translator during processing.
<CONDITION>	Global	Marks a section of an SDML file that is not processed unless one of the arguments to the tag matches the argument in the related <SET_CONDITION> tag.
<CONSTRUCT>	SOFTWARE	Specifies a variable construct and gives its expansion.
<CONSTRUCT_LIST>	SOFTWARE	Begins a list of construct items and definitions that expand on variables specified in the context of the <STATEMENT_FORMAT> tag.
<CONTENTS_FILE>	Global	Specifies that the table of contents output file for a document should be included when the document is processed.
<COPYRIGHT_DATE>	Global	Inserts a copyright date line on the copyright page along with other system-specific copyright information.
<COPYRIGHT_PAGE>	Global	Begins a copyright page and enables copyright page tags.
<CP>	Global	Marks the continuation of a paragraph that has been interrupted by another text element.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<CPAREN>	Global	Supplies an unmatched closing parenthesis in an argument to a tag.
<CPOS>	SOFTWARE	Marks the cursor position in a screen display.
<DATE>	Global	Produces the current system date or time.
<DEFINE_BOOK_NAME>	Global	Defines the title of a book and associates a user-defined symbol-name with that title for later reference.
<DEFINE_SYMBOL>	Global	Associates a string of text with a user-defined symbol, so that the text can be referenced via this symbol throughout the document.
<DEFINITION_LIST>	Global	Begins a definition list.
<DEFINITION_LIST_HEAD>	Global	Supplies the heading to precede a definition list. Output formatting is controlled by the document-type.
<DEFLIST_DEF>	Global	Begins the text that defines an item in a definition list.
<DEFLIST_ITEM>	Global	Marks an item to be defined in a definition list.
<DELAYED>	Global	Allows you to specify text that contains SDML tags in an argument to another tag and delays the execution of those contained tags.
<DELETE_KEY>	SOFTWARE	Creates a special character resembling a DELETE key on a keyboard.
<DESCRIPTION>	SOFTWARE	Begins a section of descriptive text providing detailed information on the current reference element.
<DISPLAY>	SOFTWARE	Simulates the appearance and position of characters on a terminal screen.
<DISTLIST>	LETTER	Begins a list of persons to whom you want to distribute a memo or letter.
<DOCTYPE>	Global	Specifies the document type for your file. This tag is for informational purposes only.
<DOCUMENT_ATTRIBUTES>	ARTICLE	Enables doctype-specific tags that override the default design format of the ARTICLE doctype.
<DOCUMENT_ATTRIBUTES>	REPORT	Enables doctype-specific tags that override the default design format of the REPORT doctype.
<DOCUMENT_ATTRIBUTES>	SOFTWARE	Enables doctype-specific tags that override the default design format of the SOFTWARE doctype.
<EMPHASIS>	Global	Marks a word or phrase for distinctive typographical treatment.
<ENDCOPYRIGHT_PAGE>	Global	Terminates the copyright page and optionally provides text that may be used on a document-type specific basis.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<ENDPART_PAGE>	Global	Terminates a part page and optionally specifies paging attributes for text that follows.
<ENDTITLE_PAGE>	Global	Terminates a title page and optionally specifies text to appear at the bottom of the title page.
<EXAMPLE>	Global	Labels the beginning of a formal example.
<EXAMPLES_INTRO>	SOFTWARE	Provides introductory text before an example.
<EXAMPLE_ATTRIBUTES>	Global	Specifies attributes for the current example.
<EXAMPLE_FILE>	Global	Causes a separate file containing an example to be included in the source file.
<EXAMPLE_SEQUENCE>	SOFTWARE	Begins a numbered sequence of informal examples.
<EXAMPLE_SPACE>	Global	Leaves space for an example that will be pasted in during final production.
<EXC>	SOFTWARE	Begins a code example within a series of numbered informal examples.
<EXI>	SOFTWARE	Begins an interactive example within a series of numbered informal examples.
<EXTEXT>	SOFTWARE	Terminates an example and begins an explanation in a sequence of numbered examples.
<FARG>	SOFTWARE	Adds a single argument line to a list of arguments in a routine syntax format section.
<FARGS>	SOFTWARE	Provides the argument portion of a routine syntax statement within the context of the <FORMAT> tag.
<FCMD>	Global	Specifies the keyword portion of a formatted command/parameter pair in a format section.
<FCMD>	SOFTWARE	Identifies a command or statement keyword, and an optional parameter list, within a format section.
<FFUNC>	SOFTWARE	Labels a function within the context of a <FORMAT> or <STATEMENT_FORMAT> tag section.
<FIGURE>	Global	Labels the beginning of a figure.
<FIGURE_ATTRIBUTES>	Global	Specifies the placement of a figure on the page. Also specifies valid page breaks for that figure.
<FIGURE_FILE>	Global	Includes a graphics file in your output file if the output device has graphics capability.
<FIGURE_SPACE>	Global	Marks the space required for a figure that will be pasted in during final production.
<FILE_SPEC>	Global	Allows you to use a file specification that contains angle brackets as an argument to an SDML tag without VAX DOCUMENT interpreting that file specification as a tag.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<FINAL_CLEANUP>	Global	Provides explicit formatting instructions for the text formatter to be used for final formatting and cleanup.
<FOOTNOTE>	Global	Places a footnote character in text, using the character specified in the tag's argument, and places the footnote text at the bottom of the page.
<FOOTNOTE_TEXT>	Global	Specifies the text of a footnote associated with a footnote reference.
<FOOTREF>	Global	Creates one or more footnote callouts in text or in a table using the footnote numbers or characters as arguments.
<FORMAT>	Global	Enables <FCMD> , <FPARMS> , and <FPARM> to distinguish formatted command keywords and parameters.
<FORMAT>	SOFTWARE	Begins a section that highlights the syntax of a tag, command, or routine, including keywords and arguments.
<FORMAT_SUBHEAD>	SOFTWARE	Introduces one of the multiple formats in the statement format section.
<FPARM>	Global	Specifies a parameter to be formatted following <FPARMS> , aligned under the parameter list portion of a keyword/parameter list pair.
<FPARM>	SOFTWARE	Adds a single parameter line to a list of parameters in a command or statement syntax format section.
<FPARMS>	Global	Specifies the parameter portion of a formatted command/parameter pair in a format section.
<FPARMS>	SOFTWARE	Specifies the parameters to a command or statement keyword.
<FROM_ADDRESS>	LETTER	Specifies the name and address of the sender of a letter, and places this information flush left near the right margin.
<FRONT_MATTER>	Global	Begins the front matter of a book.
<FRTN>	SOFTWARE	Specifies the routine-keyword portion of a routine syntax statement within the context of the <FORMAT> tag.
<FTAG>	SOFTWARE	Specifies the name of a tag and its arguments within the context of a <FORMAT> tag.
<FUNCTION>	SOFTWARE	Begins a new function description.
<GDEF>	Global	Begins the text that defines a term in a glossary.
<GLOSSARY>	Global	Begins a glossary of terms in a document or book.
<GRAPHIC>	SOFTWARE	Displays terminal graphics characters.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<GREF>	Global	Marks a cross-reference to a term within a glossary.
<GTERM>	Global	Labels a term to be defined in a glossary.
<HEAD>	LETTER	Creates a main heading which is placed on the left side of the page.
<HEAD>	REPORT	Creates a main heading which is placed on the left of the output page.
<HEADx>	Global	Marks a heading of the level specified (1 through 6).
<HELLIPSIS>	Global	Labels omitted material in a horizontal dotted line.
<HYPHENATE>	Global	Provides information about legal hyphenation of a word of text.
<ICON>	Global	Allows you to include a graphic image in your printed output and print text parallel to the image. The text is printed either to the right or left of the picture.
<ICON_FILE>	Global	Specifies a graphics file that accompanies text within the <ICON> and <ENDICON> tags.
<ICON_TEXT>	Global	Labels the text that accompanies a graphic image included in text with the <ICON> and <END_ICON> tags.
<INCLUDE>	Global	Causes the contents of a specified file to be included in the current input file for processing.
<INCLUDES_FILE>	Global	Equates a logical name with a file specification during processing of a profile.
<INDEX_FILE>	Global	Specifies the position in a book (or document) where an index file should be included in the output.
<INTERACTIVE>	Global	Begins an example dialog between user and system and enables the tags <S> and <U> to distinguish system text from user text.
<INTRO_SUBTITLE>	OVERHEADS	Creates a secondary title of up to four lines on an introductory slide.
<INTRO_TITLE>	OVERHEADS	Creates a main title of up to four lines on an introductory slide.
<KEEP>	Global	Specifies that a string of text should always occur on the same line of output, that is, should not be hyphenated between lines.
<KEY>	SOFTWARE	Depicts a key from a keyboard or keypad graphically.
<KEYPAD>	SOFTWARE	Specifies an individual keypad diagram and optionally supplies a title for that diagram.
<KEYPAD_ENDROW>	SOFTWARE	Displays the bottom row of an editing keypad that has up to three keys.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<KEYPAD_ROW>	SOFTWARE	Displays a row of an editing keypad that has up to four keys.
<KEYPAD_SECTION>	SOFTWARE	Begins a series of one or more keypad diagrams.
<KEYWORD>	Global	Labels a significant word that deserves to be distinguished typographically.
<KEY_NAME>	SOFTWARE	Emphasizes the name of a key within text.
<KEY_PLUS>	SOFTWARE	Creates a plus sign between keys in a key sequence example.
<KEY_SEQUENCE>	SOFTWARE	Begins a section containing one or more key representations.
<KEY_TYPE>	SOFTWARE	Provides a descriptive label for keys within a key sequence.
<LE>	Global	Labels a list element.
<LEVEL>	REPORT	Specifies an outline entry and the organizational level of that outline entry.
<LINE>	Global	Specifies that the text that follows is to be placed on a new line of output.
<LINE_ART>	Global	Labels a rough sketch produced at the terminal keyboard for draft output, to give some idea of what the final figure will look like.
<LIST>	Global	Begins a list. The type of list (for example, numbered or stacked) is specified by the argument to the <LIST> tag.
<LITERAL>	Global	Allows you to specify text that contains words in angle brackets that might otherwise be interpreted as tags.
<LOWERCASE>	Global	Labels text that should appear as lowercase in the final output.
<MARK>	Global	Indicates the beginning of new or modified information.
<MATH>	Global	Labels a short mathematical expression or the beginning of an extended mathematical example.
<MATH_CHAR>	Global	Creates a special mathematical symbol.
<MCS>	Global	Labels a character in the DEC Multinational Character Set.
<MEMO_DATE>	LETTER	Creates a line in a memo or letter that displays the date after the heading "Date:".
<MEMO_FROM>	LETTER	Identifies the name and address of the sender of a memo, and places this information flush left on the left margin, with a heading of "From:".
<MEMO_HEADER>	LETTER	Centers the heading "Interoffice Memorandum" in bold letters on the current output line.
<MEMO_LINE>	LETTER	Lets you create your own titled informational lines.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<MEMO_TO>	LETTER	Specifies the name and address of the receiver of a memo, and places this information flush left on the left margin, with a heading of "To:".
<MESSAGE_SECTION>	SOFTWARE	Begins a section of error message descriptions.
<MESSAGE_TYPE>	SOFTWARE	Establishes the format for error messages and affects the numbers of arguments passed to the <MSG> and <MSGS> tags.
<MSG>	SOFTWARE	Formats the text of a message within a series of error message descriptions.
<MSGS>	SOFTWARE	Formats the text of one or more messages within a series of error message descriptions.
<MSG_TEXT>	SOFTWARE	Labels text which describes a message in a <MESSAGE_SECTION> tag section.
<NESTED_TABLE_BREAK>	Global	Marks a place that a nested table may be broken across pages.
<NEWTERM>	Global	Labels a term first introduced into the text in order to emphasize the term. In output, the term will be italicized.
<NOTE>	Global	Labels a note, caution, warning, or some other portion of text to which you wish to draw attention.
<OPAREN>	Global	Supplies an unmatched opening parenthesis in an argument to a tag.
<ORDER_NUMBER>	Global	Labels the order number or part number that may appear on the title page of a book.
<OUTLINE>	REPORT	Begins an outline and specifies a title for the outline.
<OVERVIEW>	SOFTWARE	Provides a summary description of a reference element.
<P>	Global	Marks the beginning of a new paragraph.
<PAGE>	Global	Breaks a page of text, forcing the text that follows the tag to begin on a new page.
<PARAMDEF>	SOFTWARE	Begins the text that defines an item in a parameter definition list.
<PARAMDEFLIST>	SOFTWARE	Begins a definition list describing zero or more parameters or arguments.
<PARAMITEM>	SOFTWARE	Labels one to seven items to be defined in a parameter definition list.
<PARENDCHAR>	Global	Labels a character that will appear alone within parentheses to achieve better spacing.
<PART>	Global	Labels the start of a major division within a document, and starts it on a new page.
<PART_PAGE>	Global	Begins a divider page for a new part of a document.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<PREFACE>	Global	Labels the beginning of a preface.
<PREFACE_SECTION>	Global	Creates a major section in the preface of a book, to contain information such as a summary of changes to the book or other relevant information.
<PRINT_DATE>	Global	Inserts a print date line on the copyright page.
<PROFILE>	Global	Indicates that the source file is a profile and that a bookbuild is to be performed.
<PROMPT>	SOFTWARE	Identifies a prompt which appears on a separate line from other prompts, and any parameters associated with that prompt.
<PROMPTS>	SOFTWARE	Begins a summary of interactive prompts.
<QPAIR>	SOFTWARE	Labels a qualifier pair within a qualifier format list.
<QUALDEF>	SOFTWARE	Begins the text that defines an item in a qualifier definition list.
<QUALDEFLIST>	SOFTWARE	Begins a definition list describing zero or more command qualifiers.
<QUALITEM>	SOFTWARE	Labels one to seven items to be defined in a qualifier definition list.
<QUAL_LIST>	SOFTWARE	Begins a qualifier summary list.
<QUAL_LIST_DEFAULT_HEADS>	SOFTWARE	Modifies the default heading used by the <QUAL_LIST> tag.
<QUAL_LIST_HEADS>	SOFTWARE	Labels the headings you want to use for one or both of the columns in a qualifier format list when <QUAL_LIST> (SPECIAL) is used in unusual cases for formatting control.
<QUOTATION>	ARTICLE	Begins a quotation in which the spacing is retained and the text is not filled or justified.
<QUOTE>	Global	Labels quoted material in the output.
<REFERENCE>	Global	Makes a reference to a symbol-name in a book element or text element. When processed, <REFERENCE> is replaced with the current value of the symbol-name.
<REF_NOTE>	ARTICLE	Specifies the text of a reference note, and creates a bracketed reference number in the article text.
<REF_NOTES>	ARTICLE	Causes all accumulated reference notes to be output.
<RELATED_ITEM>	SOFTWARE	Provides a text description of a set of tags or the usage of a set of tags that may be related to the tag being described.
<RELATED_TAG>	SOFTWARE	Specifies a single tag that is related to the current tag.
<RELATED_TAGS>	SOFTWARE	Provides a summary of tags whose usage is related to the tag being described.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<RESTRICTIONS>	SOFTWARE	Labels the restrictions on the use of a reference element within a SOFTWARE reference template.
<RETTEXT>	SOFTWARE	Provides general information about the attributes of the value returned by the routine.
<RETURNS>	SOFTWARE	Provides information about the value returned by a routine.
<RETURN_VALUE>	SOFTWARE	Labels a character string return value.
<REVISION>	Global	Indicates that the document contains either new or modified information and enables the output of the <MARK> tag.
<REVISION_INFO>	Global	Labels a section on a title page that provides information on what previous books have been superseded by the current one.
<RIGHT_LINE>	Global	Specifies a line of text that is to be right-adjusted in the current text margin.
<RITEM>	SOFTWARE	Labels an item in a list of restrictions.
<ROUTINE>	SOFTWARE	Begins a new routine description.
<ROUTINE_SECTION>	SOFTWARE	Begins a routine reference section, enables tags reserved for use in routine sections, and sets paging attributes.
<RSDEFLIST>	SOFTWARE	Begins a return status definition list in the routine template.
<RSITEM>	SOFTWARE	Specifies the return status value of a routine and lists its meaning.
<RULE>	Global	Outputs a rule within a table.
<RUNNING_FEET>	ARTICLE	Creates a heading at the bottom of each page.
<RUNNING_FEET>	OVERHEADS	Specifies text to be placed at the bottom of the next slide and all subsequent slides.
<RUNNING_FEET>	REPORT	Creates a heading at the bottom of each page.
<RUNNING_FEET>	SOFTWARE.SPEC	Creates a heading at the bottom of each page.
<RUNNING_TITLE>	ARTICLE	Creates a one or two line running heading at the top of each page.
<RUNNING_TITLE>	OVERHEADS	Specifies text to be placed at the top of the next slide and all subsequent slides.
<RUNNING_TITLE>	REPORT	Creates a one or two line running heading at the top of each page.
<RUNNING_TITLE>	SOFTWARE.SPEC	Creates a one or two line running heading at the top of each page.
<S>	Global	Labels the system portion of a dialog between user and system in an interactive example.
<SALUTATION>	LETTER	Specifies the salutation for the letter.
<SAMPLE_TEXT>	Global	Distinguishes, typographically, an extract of text.
<SDML_TAG>	SOFTWARE	Begins a new tag description.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<SECTION>	REPORT	Begins a new page and creates a major heading on the left side of that page.
<SET_APPENDIX_LETTER>	Global	Overrides the default appendix letter assigned to an appendix by VAX DOCUMENT.
<SET_APPENDIX_NUMBER>	MILSPEC	Overrides the default appendix roman number assigned to an appendix by VAX DOCUMENT.
<SET_CHAPTER_NUMBER>	Global	Overrides the default chapter number assigned to a chapter by VAX DOCUMENT.
<SET_CONDITION>	Global	Creates or removes a condition-name.
<SET_FIGURE_FILE_SPACING_ DEFAULT>	Global	Overrides the default amount of vertical blank space that appears before and after an included graphics file.
<SET_TABLE_ROW_BREAK_ DEFAULT>	Global	Overrides the default value for a multipage table's first valid break.
<SET_TEMPLATE_COMMAND>	SOFTWARE	Defines a new tag with the same function as the <COMMAND> tag, and changes the format of command descriptions produced using the new tag.
<SET_TEMPLATE_HEADING>	SOFTWARE	Overrides the heading for all subsequent uses of a template tag.
<SET_TEMPLATE_LIST>	SOFTWARE	Creates a user-defined set of tags for listing information.
<SET_TEMPLATE_PARA>	SOFTWARE	Defines a set of template tags for setting the format of a paragraph of information.
<SET_TEMPLATE_ROUTINE>	SOFTWARE	Defines a new reference element tag name to use in the routine template, and specifies the formatting attributes for the new tag.
<SET_TEMPLATE_ STATEMENT>	SOFTWARE	Defines a new reference element tag name to use in the statement template, and specifies the formatting attributes for the new tag.
<SET_TEMPLATE_ SUBCOMMAND>	SOFTWARE	Changes the name of the <SUBCOMMAND> tag to the name you specify, and specifies formatting attributes for the new tag.
<SET_TEMPLATE_TABLE>	SOFTWARE	Defines a set of template tags for setting information in two- or three-column lists.
<SET_TEMPLATE_TAG>	SOFTWARE	Defines a new reference element tag name to use in the tag template, and specifies formatting attributes for the newly-defined tag.
<SHOW_LEVELS>	REPORT	Emphasizes text within an outline.
<SIGNATURES>	REPORT	Begins a list of signatures which are to appear in the front matter of a document.
<SIGNATURES>	SOFTWARE	Begins a list of signatures which are to appear in the front matter of a document.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<SIGNATURE_LINE>	MILSPEC	Creates up to two rules on a line and places a name below each rule; each rule is used as a signatory line for the person listed below it.
<SIGNATURE_LIST>	MILSPEC	Begins a two-column listing of signature lines on the title page of a document and supplies headings for each of those columns.
<SINGLE_QUOTE>	Global	Outputs, within text, a single quotation mark as it appears on a keyboard.
<SLIDE>	OVERHEADS	Begins a new overhead slide.
<SOURCE_NOTE>	ARTICLE	Provides information pertaining to the original source of information for an article.
<SPECIAL_CHAR>	Global	Provides access to special characters that are not available on the terminal keyboard.
<SPECIFICATION_INFO>	MILSPEC	Creates a listing of information about the specification on the title page and creates a two-line running heading for the rest of the document.
<SPEC_TITLE>	MILSPEC	Creates a title with up to seven centered lines on the title page.
<STATEMENT>	SOFTWARE	Begins a new statement description.
<STATEMENT_FORMAT>	SOFTWARE	Begins a section that illustrates the syntax of a statement or function including keywords and parameters.
<STATEMENT_LINE>	SOFTWARE	Indicates the position of a valid statement line within the context of a statement format or a construct list.
<STATEMENT_SECTION>	SOFTWARE	Begins a statement reference section, enables tags reserved for use in statement sections, and sets paging attributes.
<SUBCOMMAND>	SOFTWARE	Begins a new subcommand description.
<SUBCOMMAND_SECTION>	SOFTWARE	Begins a subcommand reference section within the command section. Use this section of a reference document for subordinate commands.
<SUBCOMMAND_SECTION_HEAD>	SOFTWARE	Specifies the heading for text that precedes a subcommand section.
<SUBHEADx>	Global	Marks an unnumbered subsidiary heading.
<SUBJECT>	LETTER	Specifies the subject of a memo or letter and places this information with a heading of "Subject:" at the left margin.
<SUBTITLE>	ARTICLE	Specifies a subtitle for an article.
<SUBTITLE>	MILSPEC	Creates a subtitle with up to seven centered lines on the title page.
<SUBTITLE>	OVERHEADS	Specifies a secondary title line for a new slide.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<SYNTAX>	SOFTWARE	Labels a statement that illustrates the syntax of a programming language.
<SYNTAX_DEFAULT_HEAD>	SOFTWARE	Creates a default heading for the <SYNTAX> tag.
<TABLE>	Global	Begins a sequence of columnar data.
<TABLE_ATTRIBUTES>	Global	Requests special formatting.
<TABLE_FILE>	Global	Causes a separate file containing a formal table to be included in the SDML input file.
<TABLE_HEADS>	Global	Specifies column headings for each column in the table.
<TABLE_KEY>	Global	Begins a key or legend for a table.
<TABLE_KEYREF>	Global	Specifies that a table key should be printed below the table (or portion of the table) in which this tag appears.
<TABLE_ROW>	Global	Specifies text for each column in the current table.
<TABLE_ROW_BREAK>	Global	Specifies the boundaries within which a long table can be broken onto a new page.
<TABLE_SETUP>	Global	Declares the number of columns in a table and the effective (approximate) width values to be assigned to each column.
<TABLE_SPACE>	Global	Marks the space required for a table that will be pasted in during final production.
<TABLE_UNIT>	Global	Begins a portion of a table containing rows that are to be grouped as logical units.
<TABLE_UNIT_HEADS>	Global	Specifies headings to be used for a table unit.
<TAG>	Global	Labels a tag.
<TAG_SECTION>	SOFTWARE	Begins a tag reference section, enables tags reserved for use in tag sections, and sets paging attributes.
<TERMINATING_TAG>	SOFTWARE	Specifies the required terminator for a tag.
<TEXT_SIZE>	OVERHEADS	Changes the size of type used within the context of topics, tables, and lists.
<TITLE>	ARTICLE	Specifies the main title line for an article.
<TITLE>	Global	Labels the title used on either a title page or part page.
<TITLE>	OVERHEADS	Specifies a title line for a new slide.
<TITLE_PAGE>	Global	Labels the beginning of a title page and enables the title page tags.
<TITLE_SECTION>	ARTICLE	Begins the title section of an article which spans both columns of the article.
<TOPIC>	OVERHEADS	Specifies a line of topic text for a slide.

Table D–1 (Cont.) Summary of VAX DOCUMENT Tags

Tag	Doctype	Description
<TO_ADDRESS>	LETTER	Identifies the name and address of the receiver of a letter and places this information flush left on the left margin.
<U>	Global	Labels the user portion of a dialog between user and system in an interactive example.
<UNDERLINE>	Global	Marks a portion of text to be underlined.
<UPDATE_RANGE>	Global	Marks the location at which a new section of updated pages begins.
<UPPERCASE>	Global	Labels text that should appear as uppercase in the final output.
<USER_I_MESSAGE>	Global	Sends an informational message to the terminal or log file during processing of a file.
<USER_W_MESSAGE>	Global	Sends a warning message to the terminal or log file during processing of a file.
<VALID_BREAK>	Global	Labels a permissible page break within a monospaced example.
<VALID_TABLE_ROW_BREAK>	Global	Marks a permissible place that a first-level table row may be broken across pages.
<VARIABLE>	Global	Labels a program variable or number.
<VBAR>	Global	Labels an occurrence of a vertical bar in an argument to a tag.
<VITA>	ARTICLE	Provides information about the author's professional history.
<X>	Global	Creates an index entry with a reference to the page on which this tag appears.
<Y>	Global	Creates an index entry with no reference to the page on which this tag appears. Used for cross-references ("See" or "See also" entries).

Index

D

E

P

Q

R

S

T

U

V

X

Y

Reader's Comments

VAX DOCUMENT
User Manual, Volume 1
AA–JT84B–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

What I like best about this manual is ______________________________

What I like least about this manual is ______________________________

I found the following errors in this manual:

Page Description

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. __________

Company ______________________________ Date __________

Mailing Address ______________________________

______________________________ Phone __________

- Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

– Do Not Tear - Fold Here